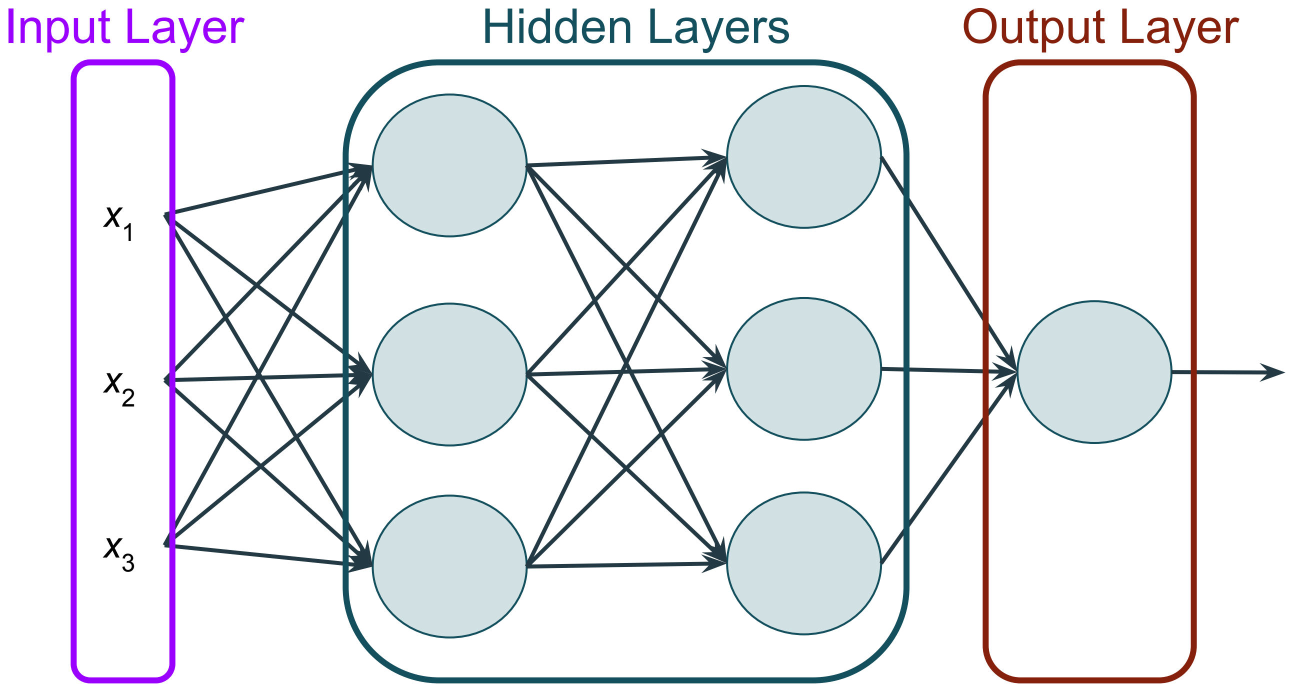
Advanced Neural Networks

CS344
Deep Learning

WELLESLEY

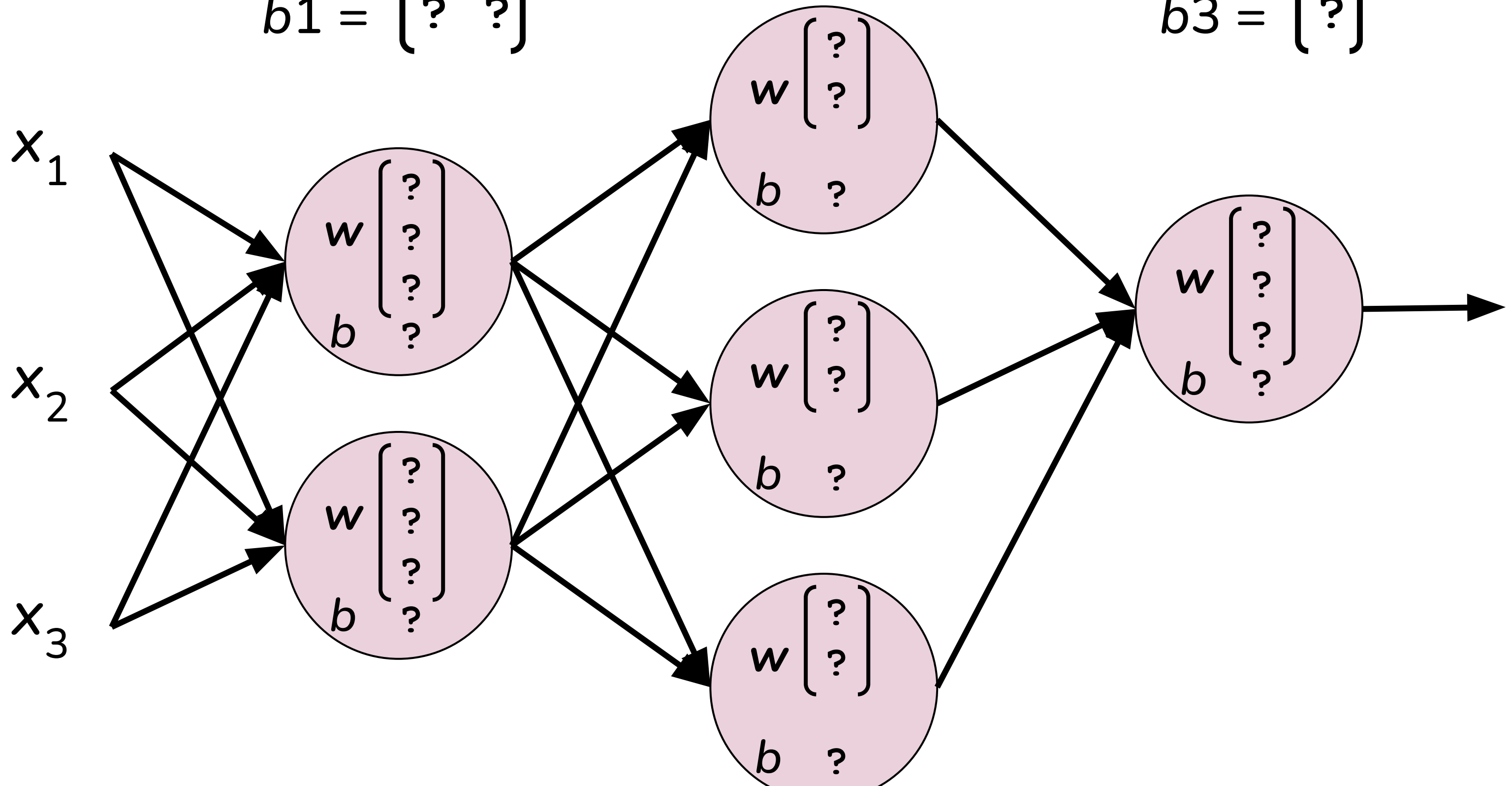
Deep Learning

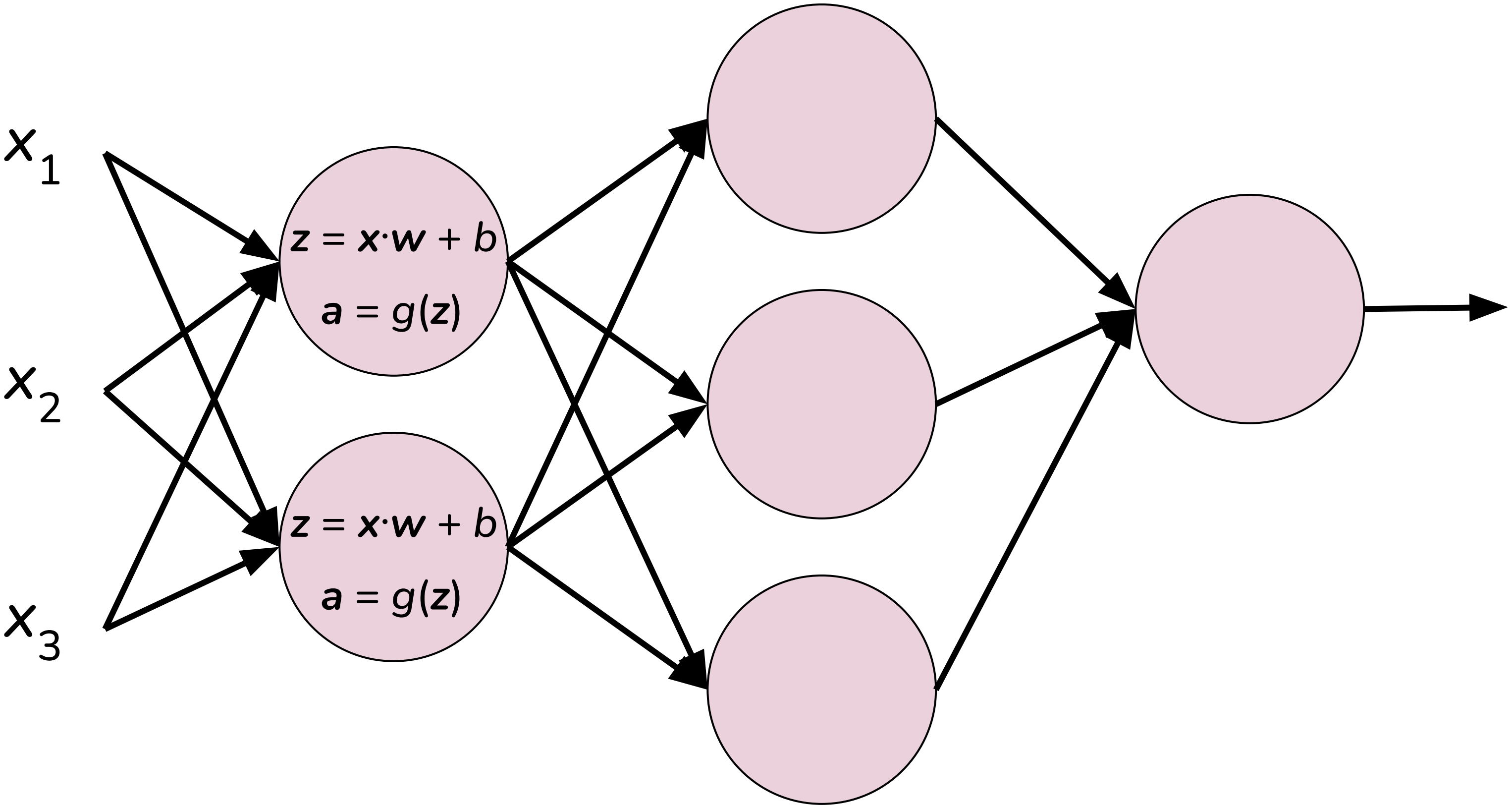


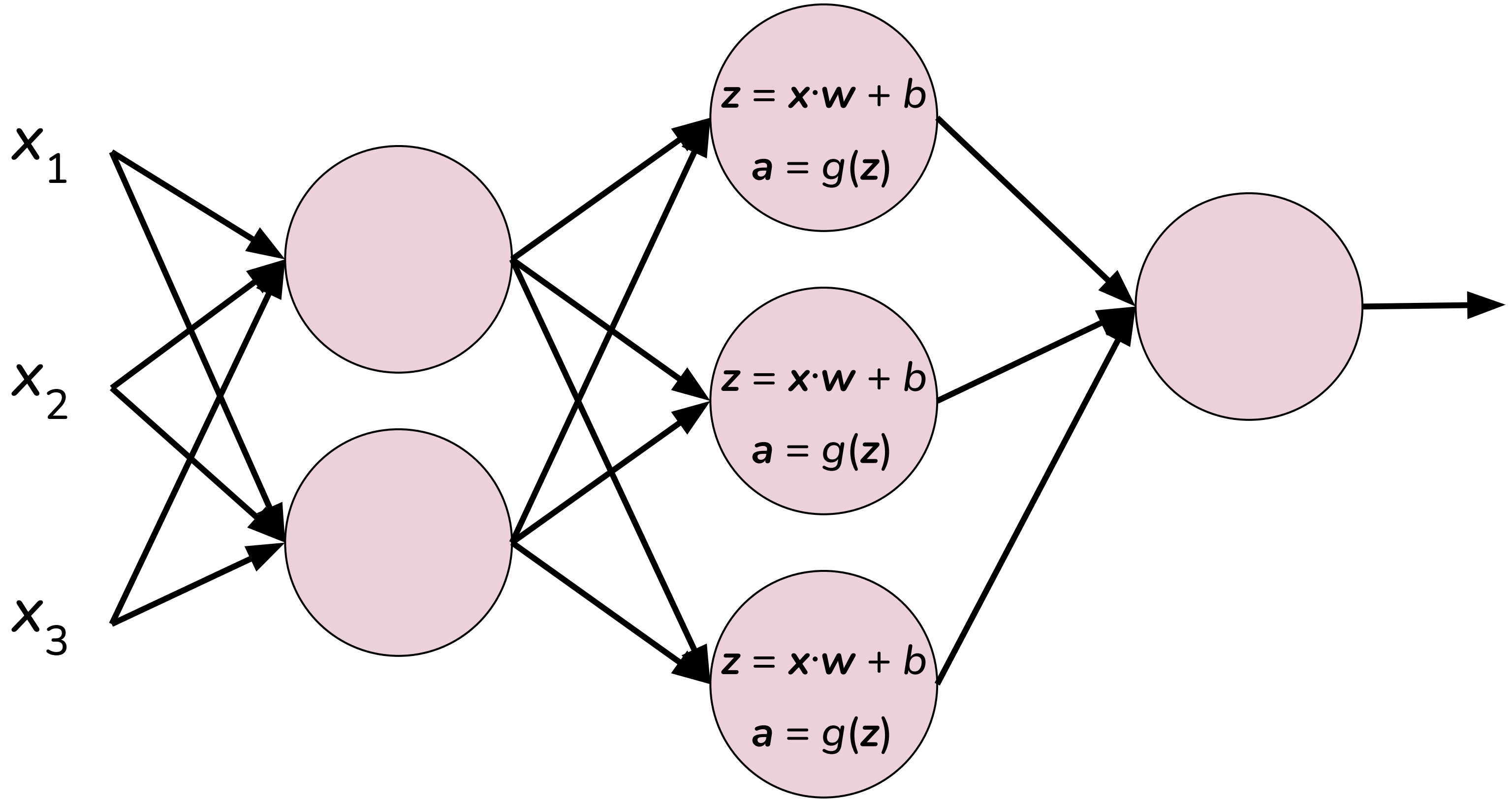
Hidden layers enable NNs to learn their own non-linear features!

$$W1 = \begin{pmatrix} ? & ? \\ ? & ? \\ ? & ? \end{pmatrix} \qquad W2 = \begin{pmatrix} ? & ? & ? \\ ? & ? & ? \end{pmatrix} \qquad W3 = \begin{pmatrix} ? \\ ? \\ ? \end{pmatrix}$$

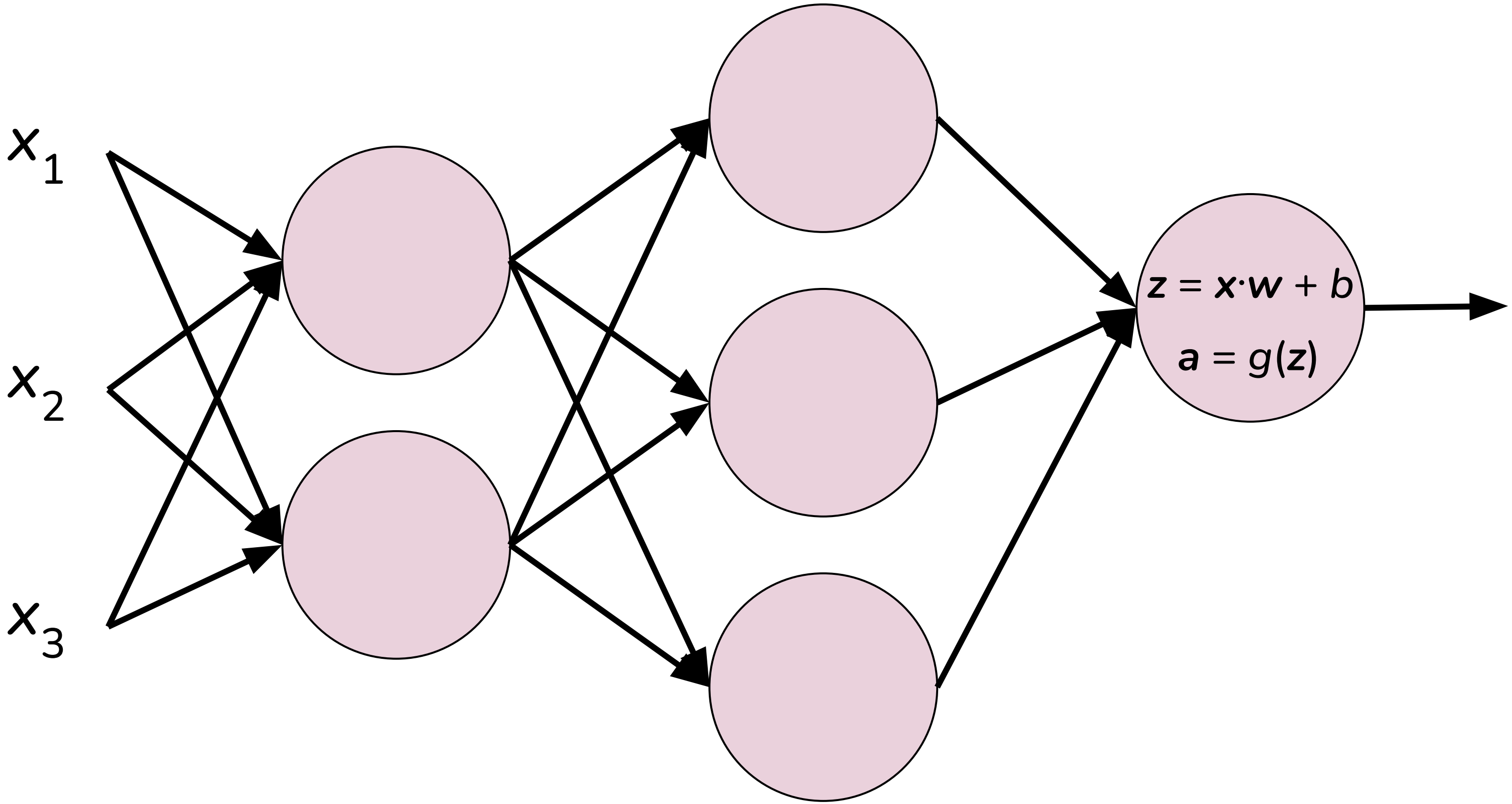
$$b1 = \begin{pmatrix} ? & ? \end{pmatrix} \qquad b2 = \begin{pmatrix} ? & ? & ? \end{pmatrix} \qquad b3 = \begin{pmatrix} ? \end{pmatrix}$$




x_1
x_2
x_3
$z = x \cdot w + b$
$a = g(z)$
$z = x \cdot w + b$
$a = g(z)$



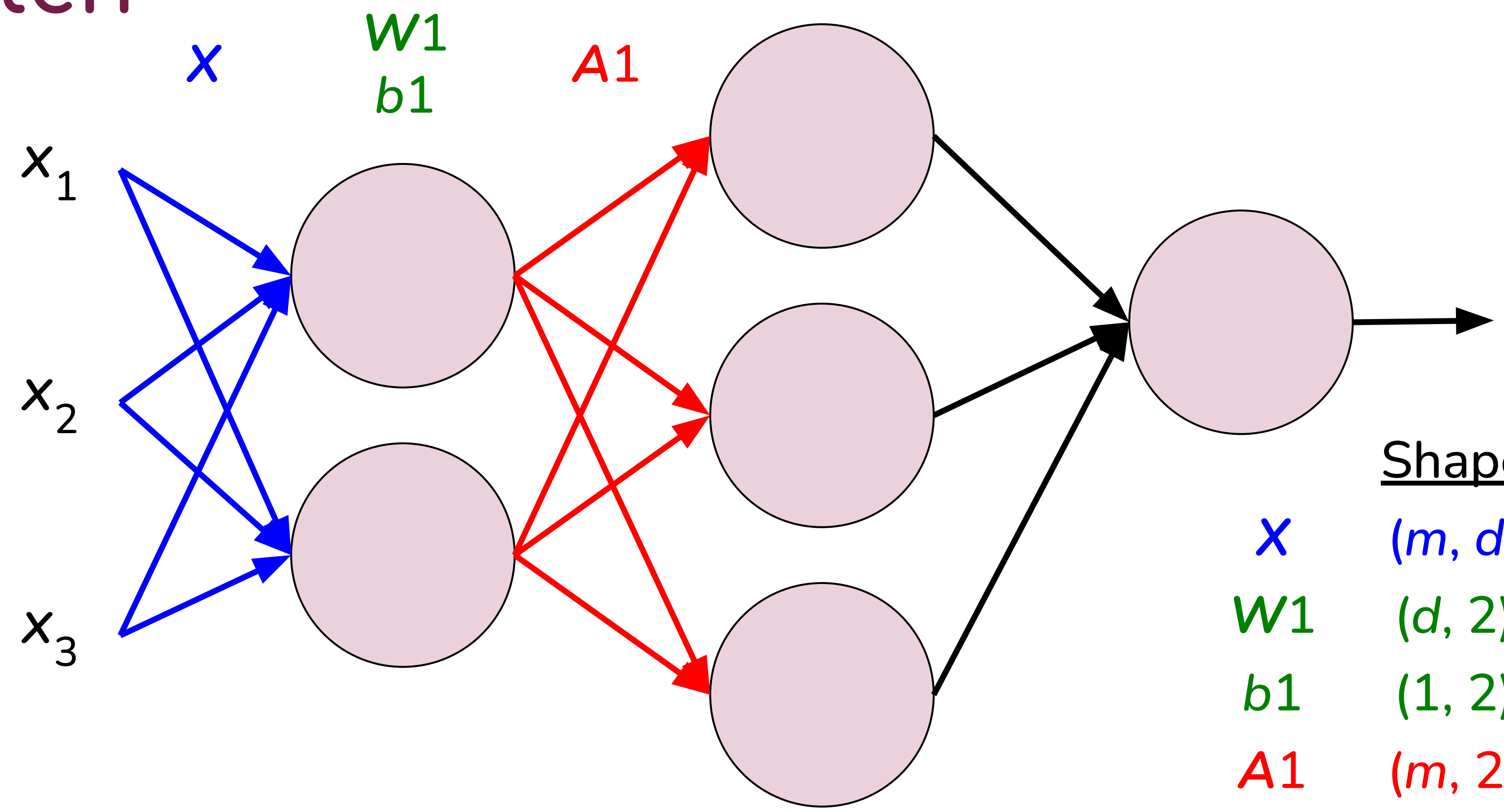
x_1
x_2
x_3
$z = \boldsymbol{x} \cdot \boldsymbol{w} + b$
$\boldsymbol{a} = g(\boldsymbol{z})$
$z = \boldsymbol{x} \cdot \boldsymbol{w} + b$
$\boldsymbol{a} = g(\boldsymbol{z})$
$z = \boldsymbol{x} \cdot \boldsymbol{w} + b$
$\boldsymbol{a} = g(\boldsymbol{z})$



x_1
x_2
x_3
$\boldsymbol{z} = \boldsymbol{x} \cdot \boldsymbol{w} + b$
$\boldsymbol{a} = g(\boldsymbol{z})$

Forward Propagation: Batch

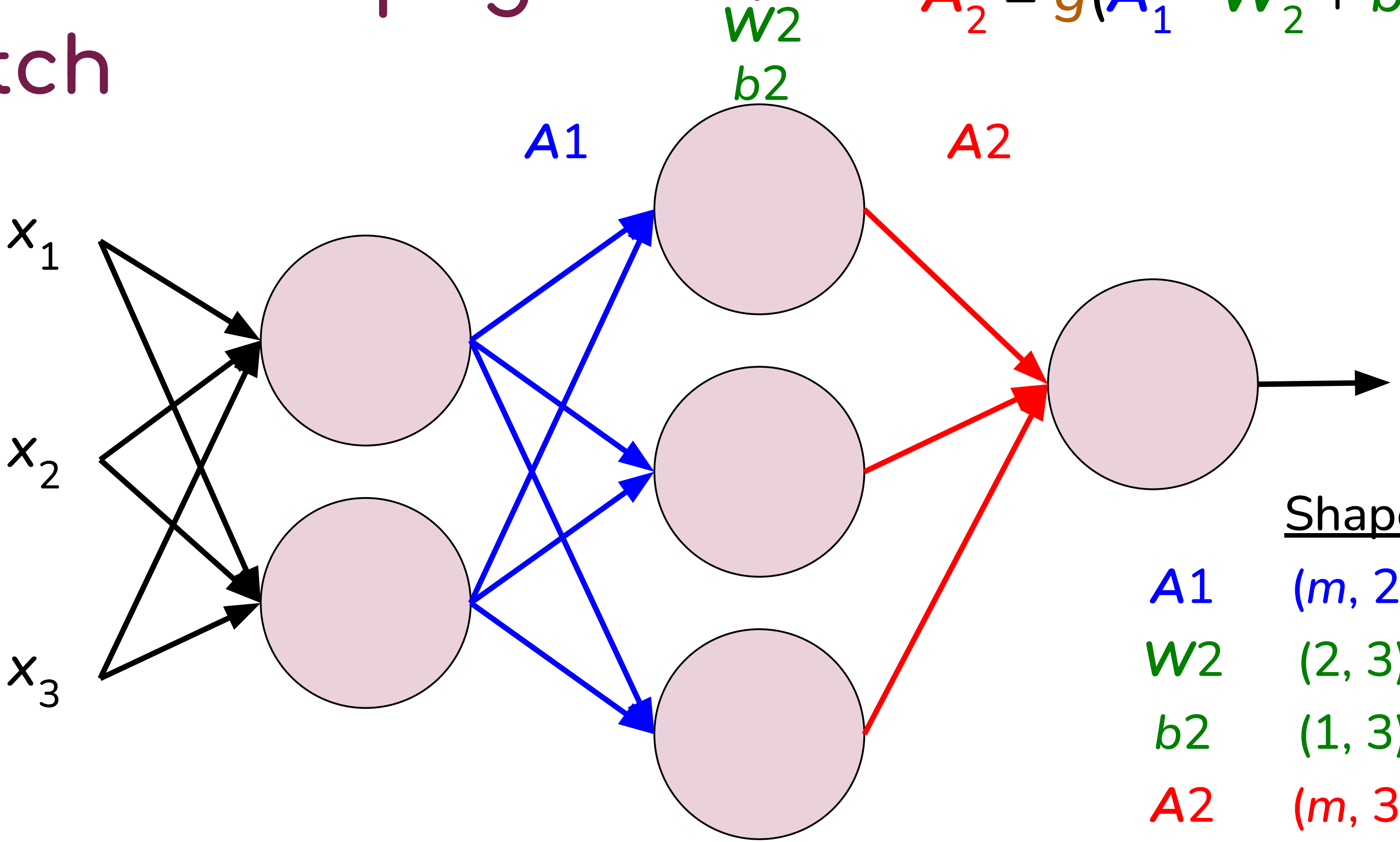
$$A_1 = g(X \cdot W_1 + b_1)$$



Shape	
X	(m, d)
$W1$	$(d, 2)$
$b1$	$(1, 2)$
$A1$	$(m, 2)$

Forward Propagation: Batch

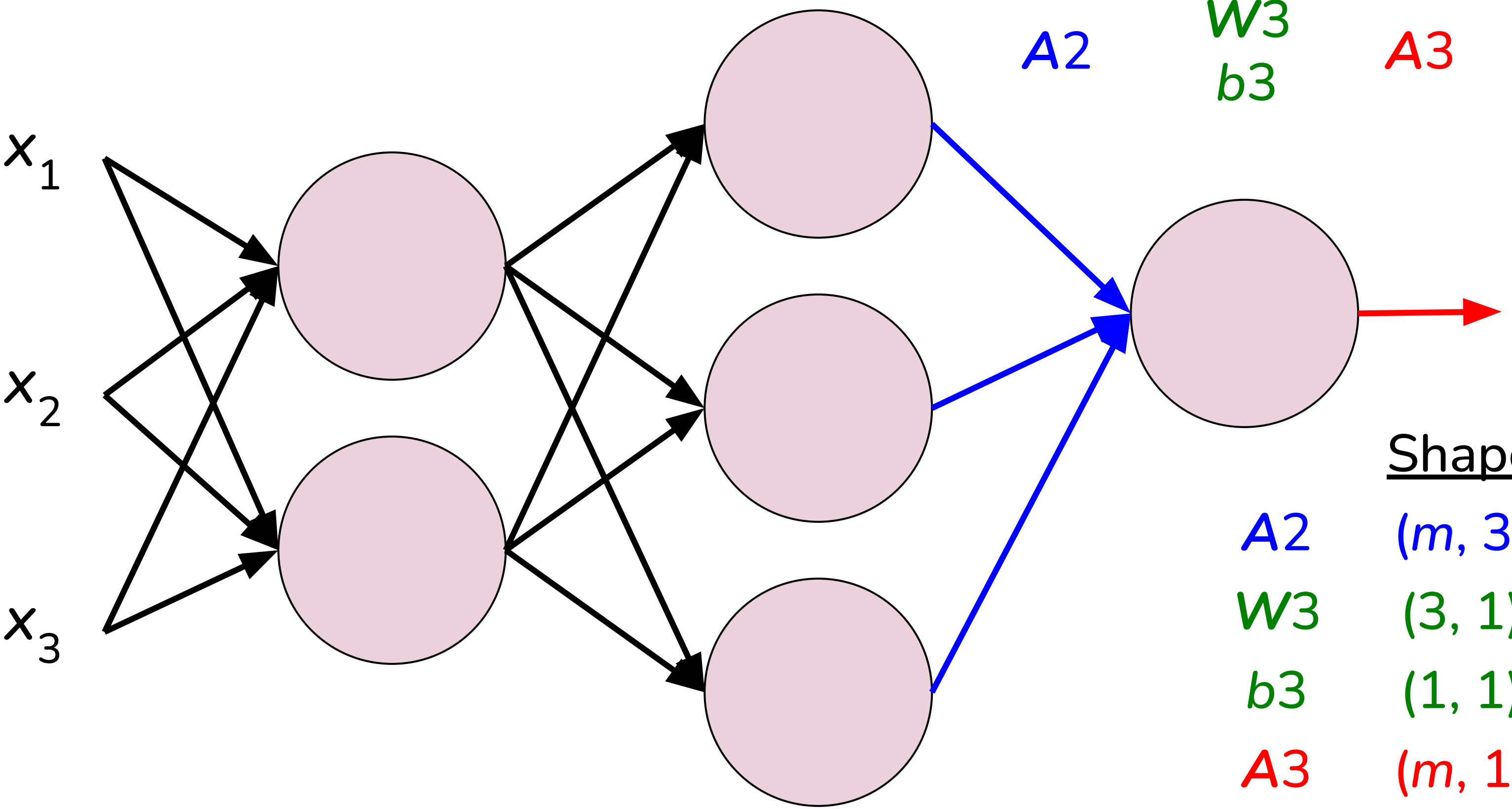
$$A_2 = g(A_1 \cdot W_2 + b_2)$$



Shape	
A1	$(m, 2)$
W2	$(2, 3)$
b2	$(1, 3)$
A2	$(m, 3)$

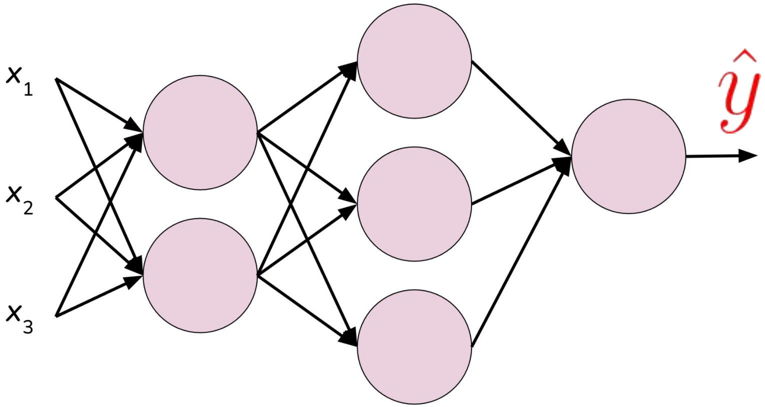
Forward Propagation: Batch

$$A_3 = g(A_2 \cdot W_3 + b_3)$$



Shape	
A2	$(m, 3)$
W3	$(3, 1)$
b3	$(1, 1)$
A3	$(m, 1)$

Loss and Cost



The *loss function*, L, quantifies the error, i.e., how far our prediction $\hat{y}$ is from the true label y

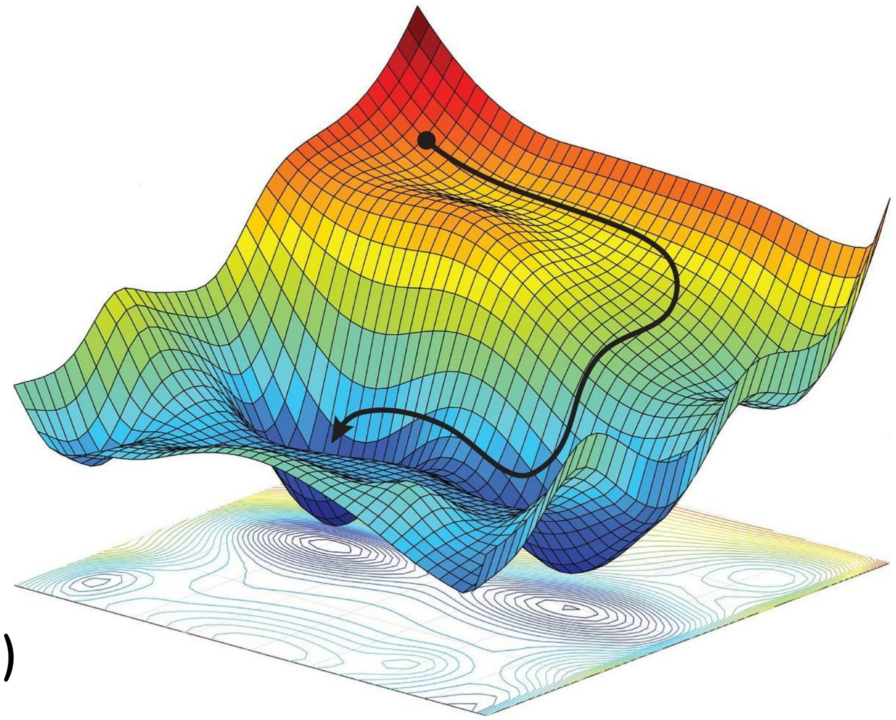
$$L = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

The *cost function*, J, is the average loss (error) over all data points

$$J = \frac{1}{m}\sum_{i=1}^{m} L \;= \frac{1}{m}\sum_{i=1}^{m} -y_i\log(\hat{y}_i) - (1-y_i)\log(1-\hat{y}_i)$$

Training

We want to find parameters (***W***’s and *b*’s) that minimize the cost, J



<u>Gradient Descent Algorithm</u>

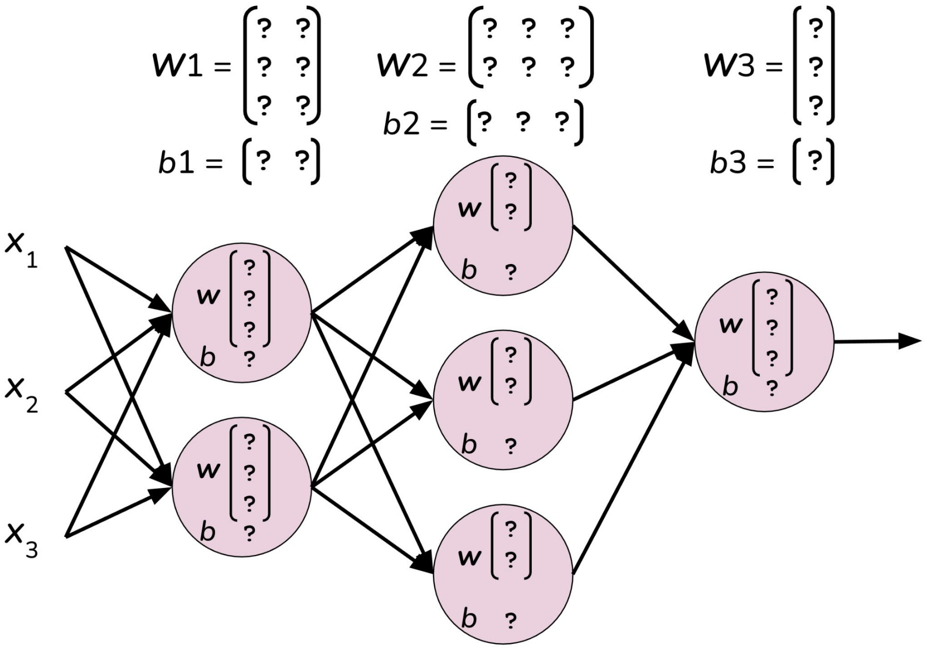
❖ Initialize parameters (***W***’s and *b*’s)

❖ Repeat until converge:
  ➢ Update parameters (***W***’s and *b*’s) to reduce the cost, J

Random Initialization

- If units in the same layer *start* with the same parameters then they will *end* with the same parameters

- There's no point in having repetitive units, i.e., multiple units in a layer with the same parameter values



- Thus, we initialize the parameters *randomly* so they start (and end) with different values

Gradient Descent

- Initialize parameters (***W***'s and *b*'s)

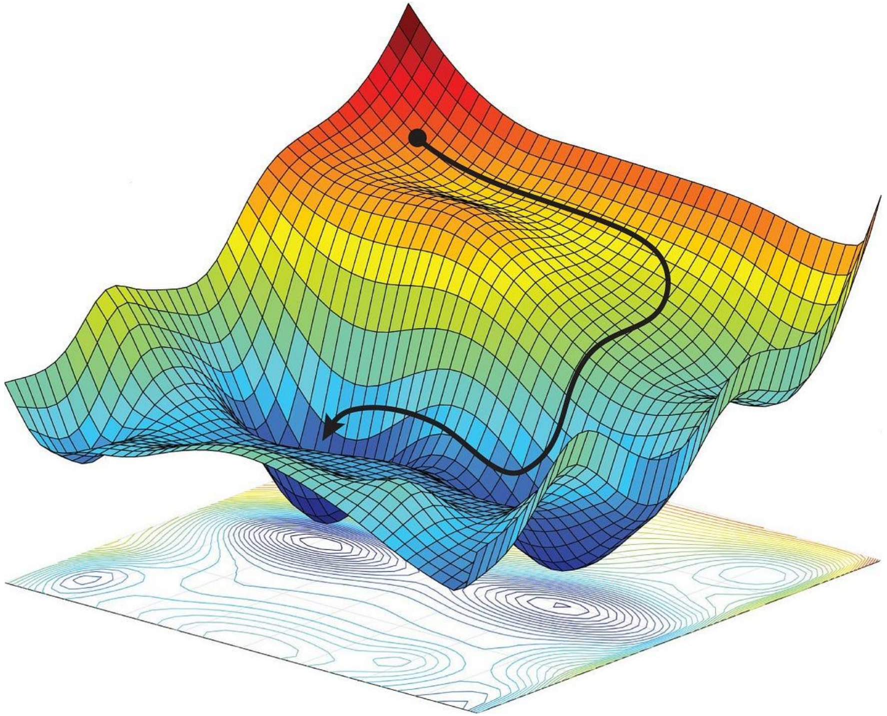
- Repeat until converge:

$$\boldsymbol{W}2 = \boldsymbol{W}2 - \alpha d\boldsymbol{W}2$$

$$b2 = b2 - \alpha db2$$

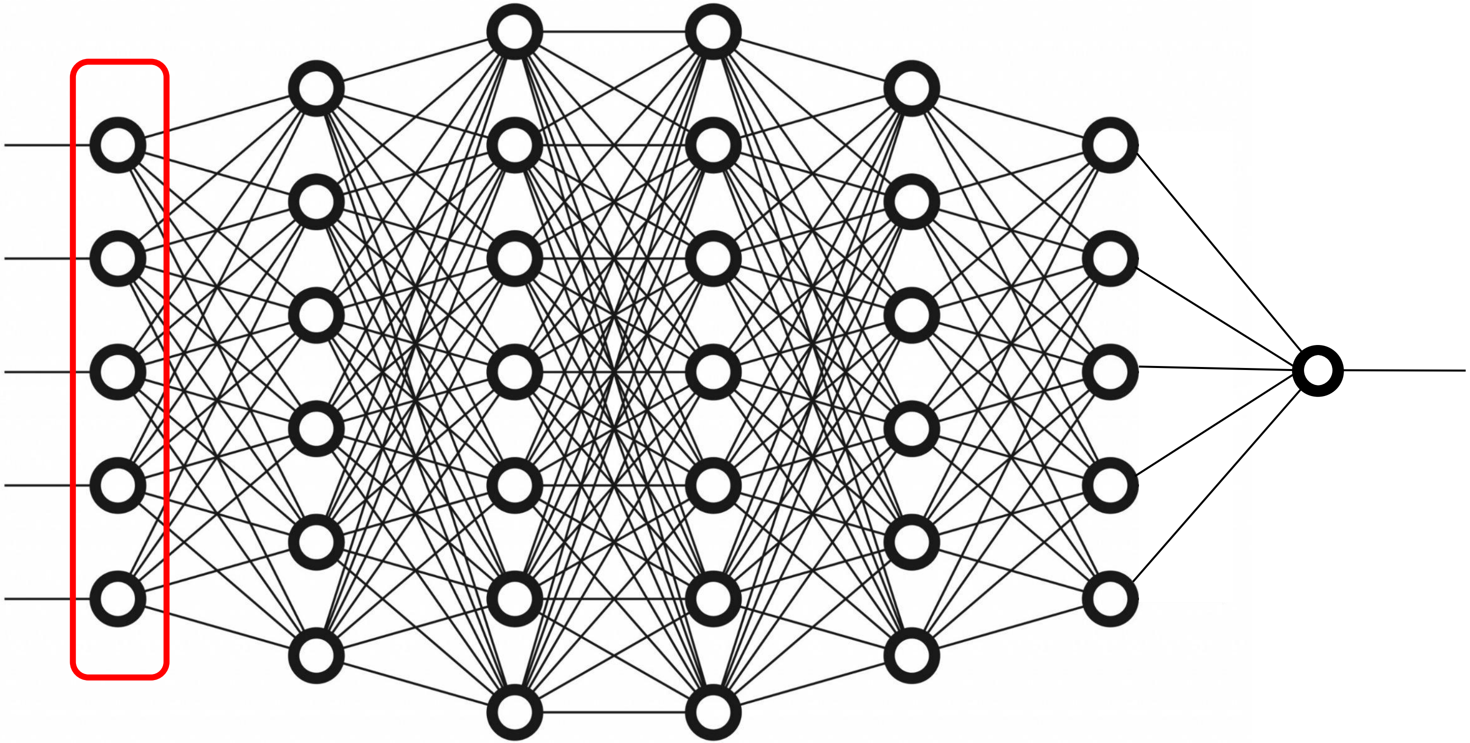
$$\boldsymbol{W}1 = \boldsymbol{W}1 - \alpha d\boldsymbol{W}1$$

$$b1 = b1 - \alpha db1$$



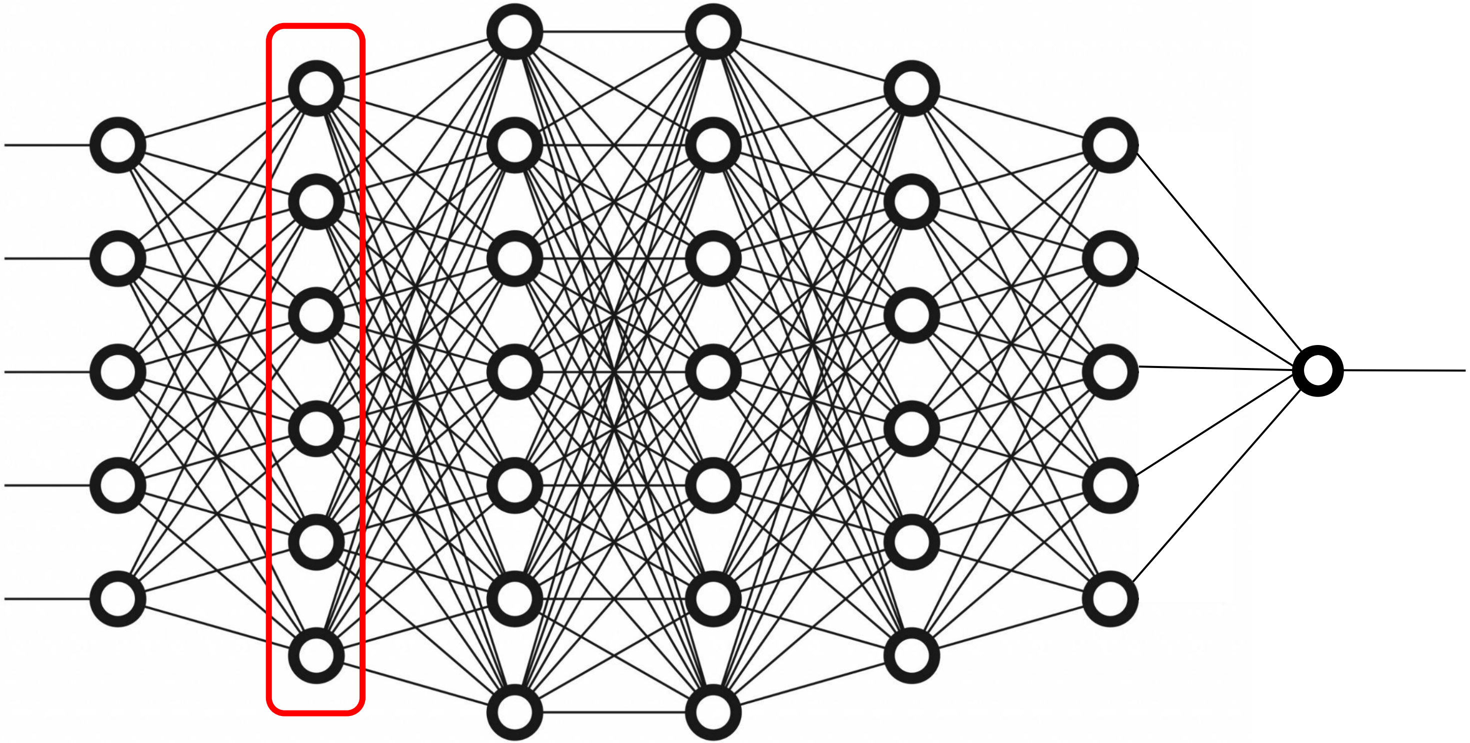
Deep Networks

Forward propagation



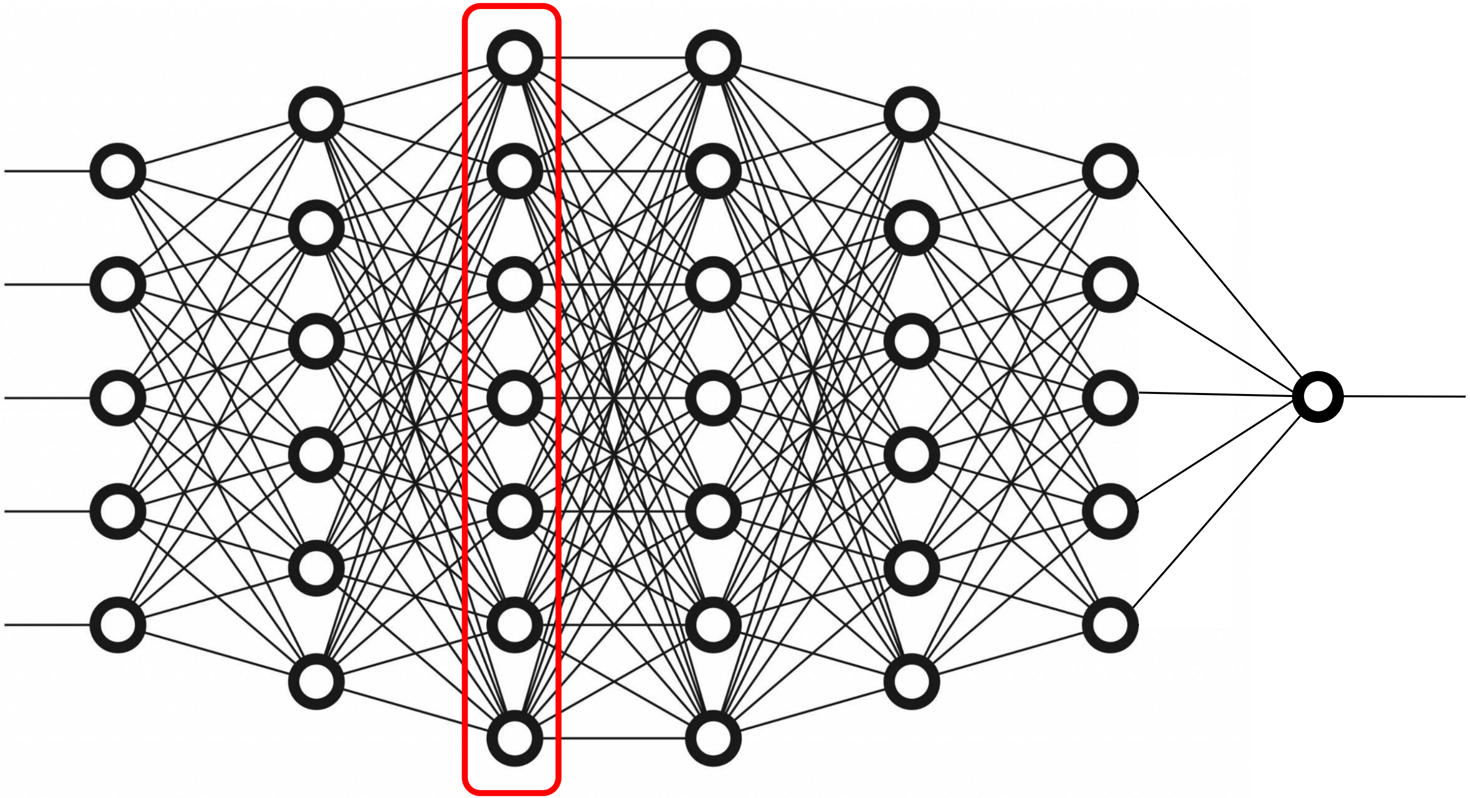
Deep Networks

Forward propagation



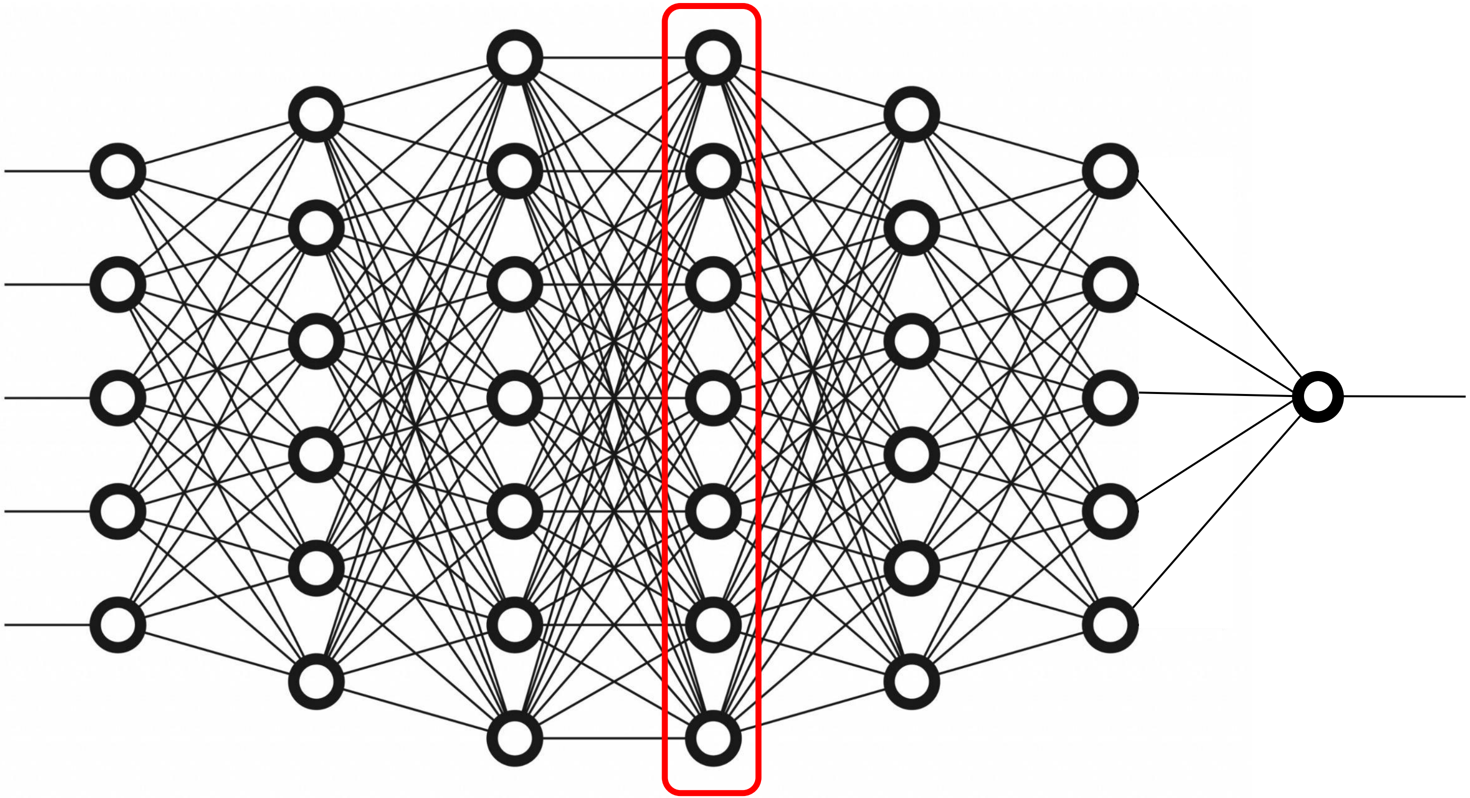
Deep Networks

Forward propagation



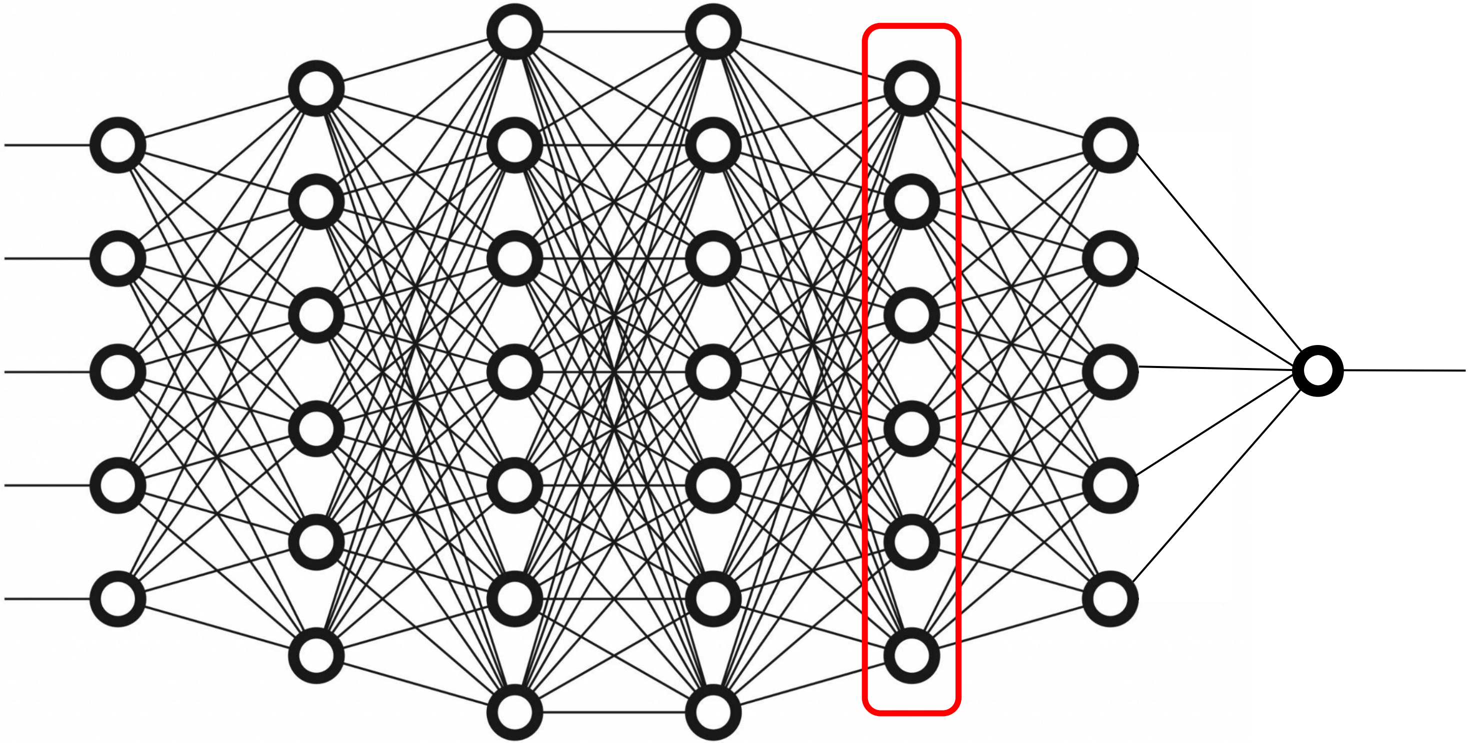
Deep Networks

Forward propagation



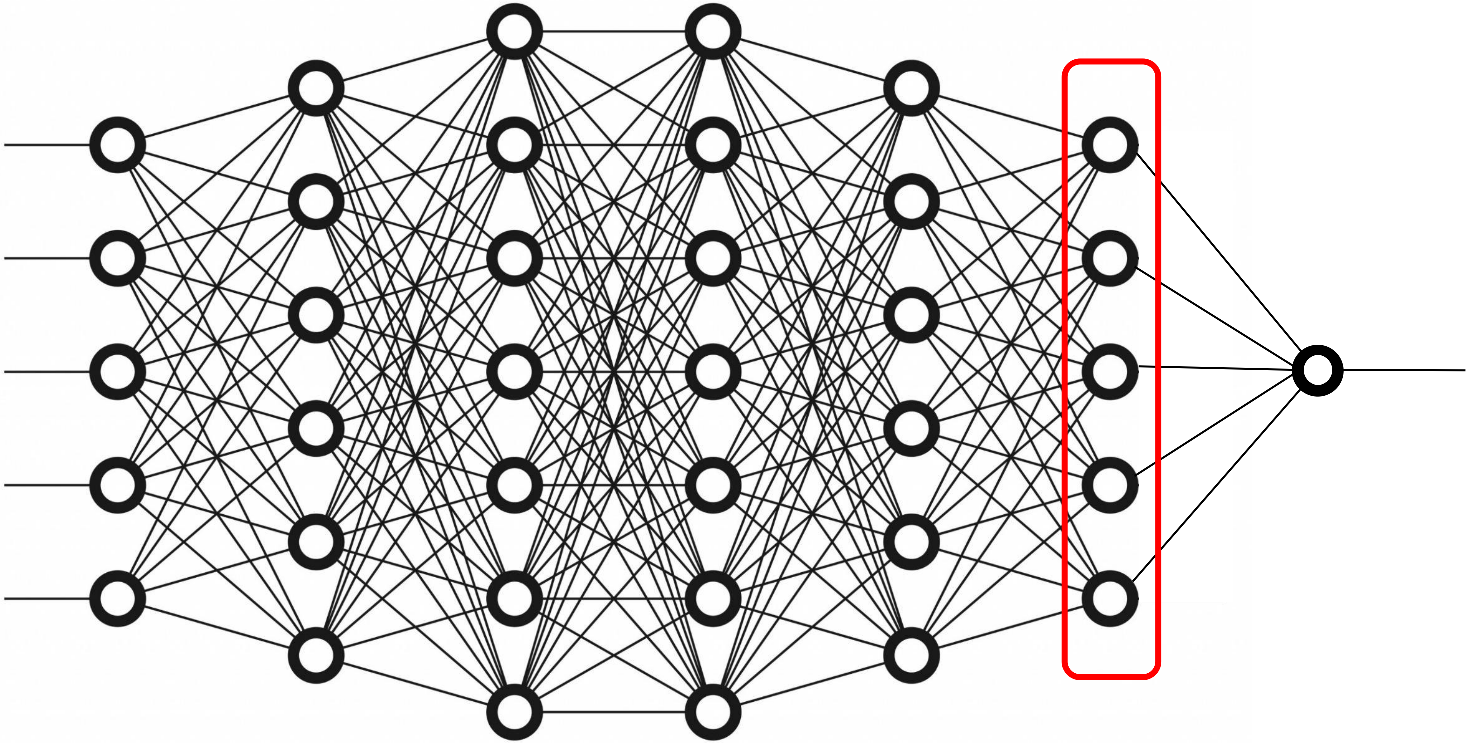
Deep Networks

Forward propagation



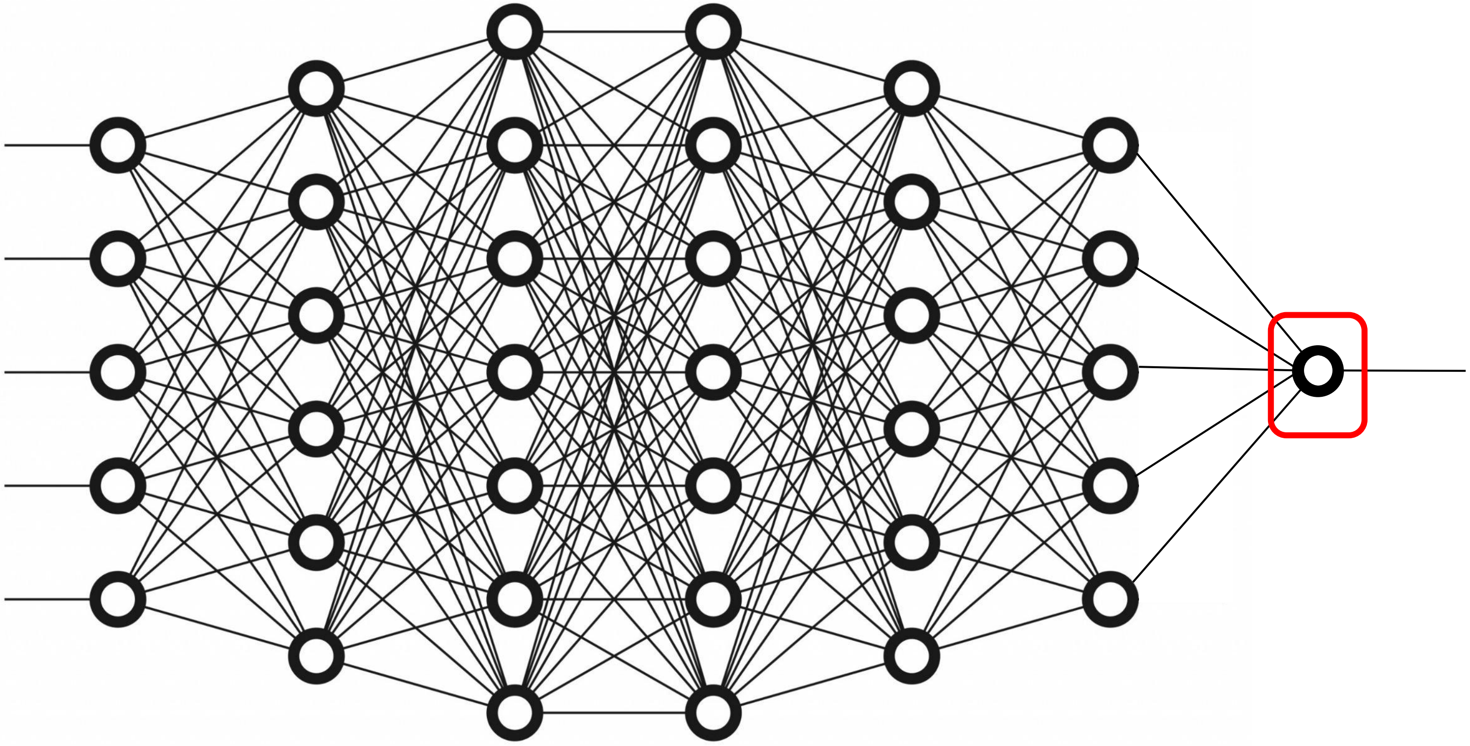
Deep Networks

Forward propagation



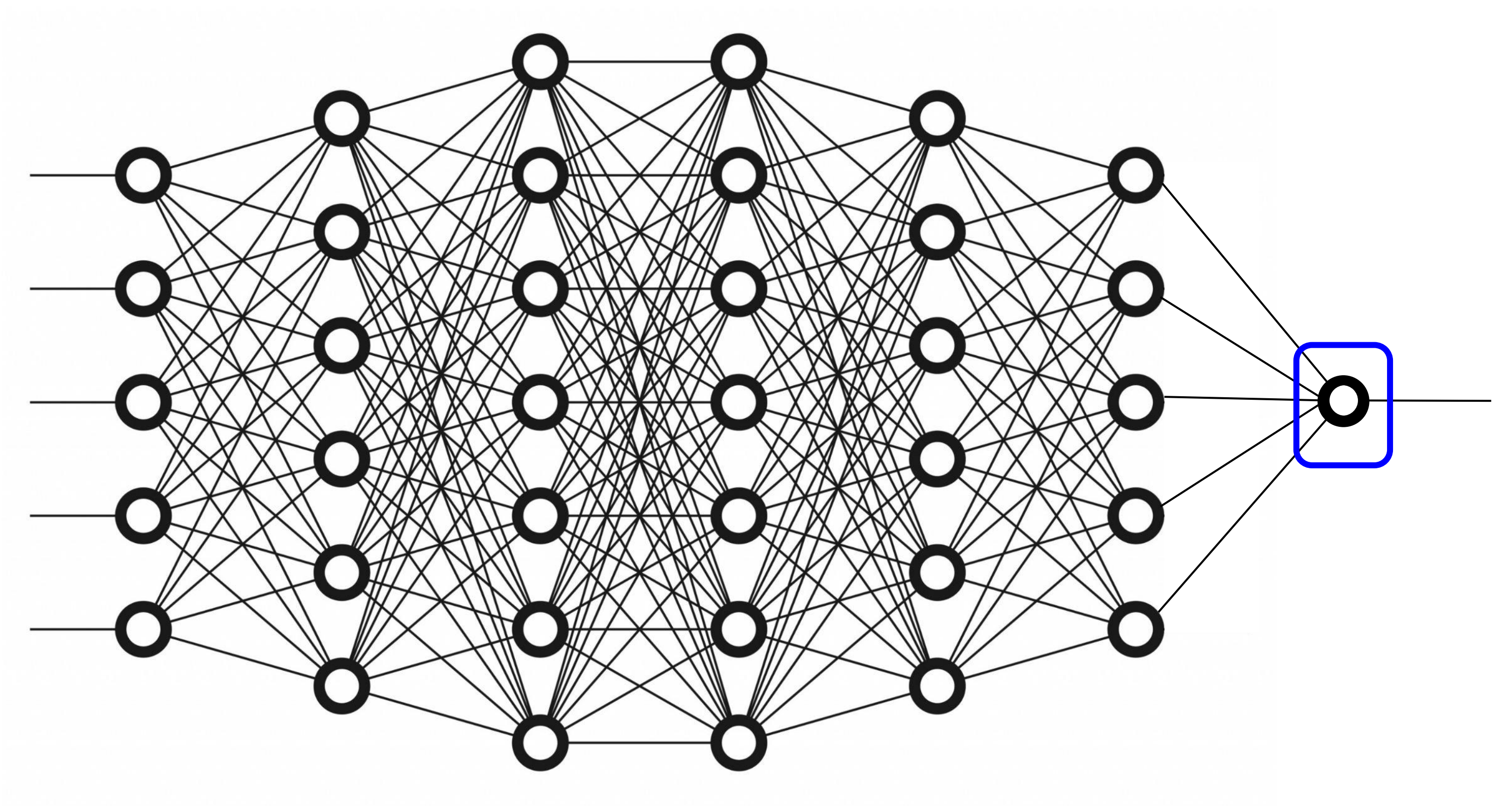
Deep Networks

Forward propagation



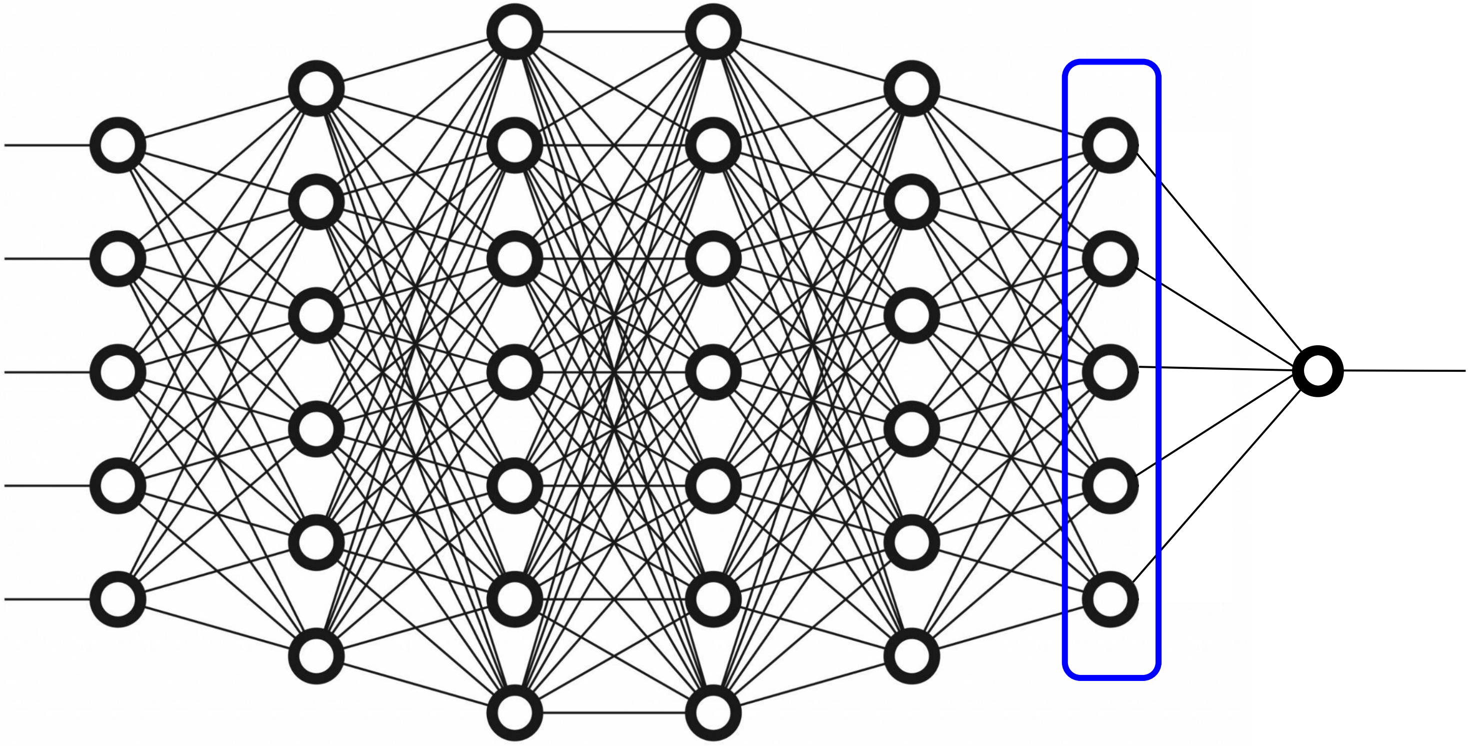
Deep Networks

Backward propagation



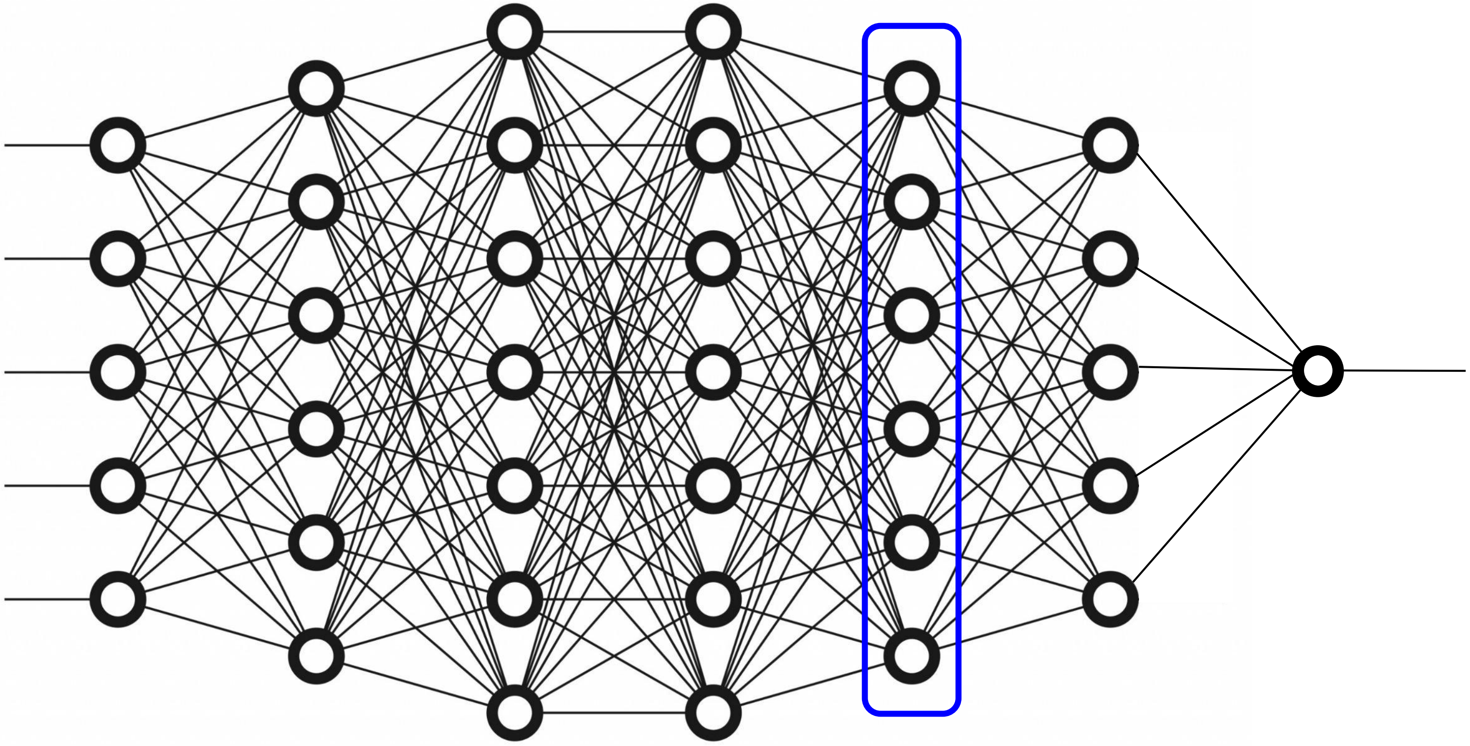
Deep Networks

Backward propagation



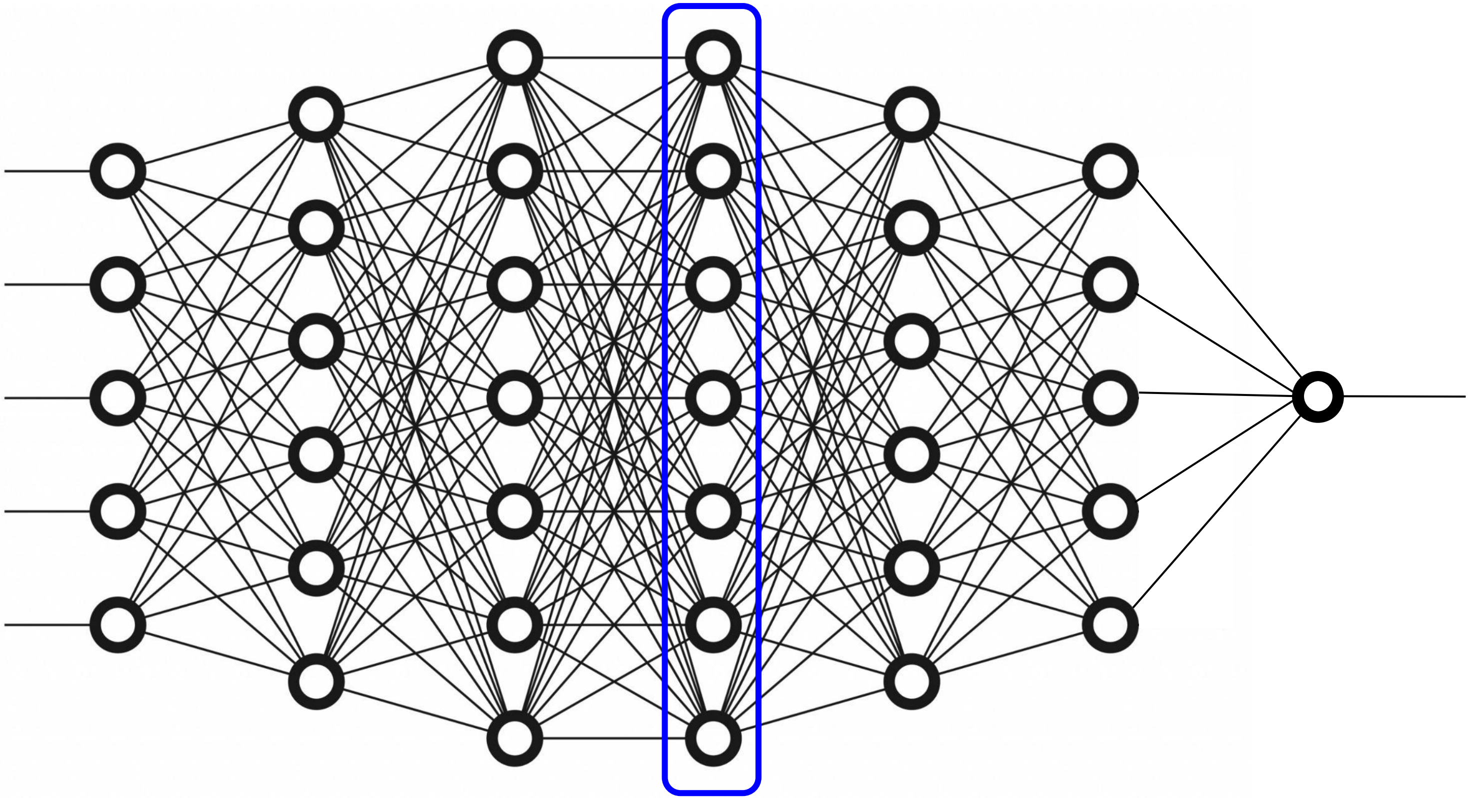
Deep Networks

Backward propagation



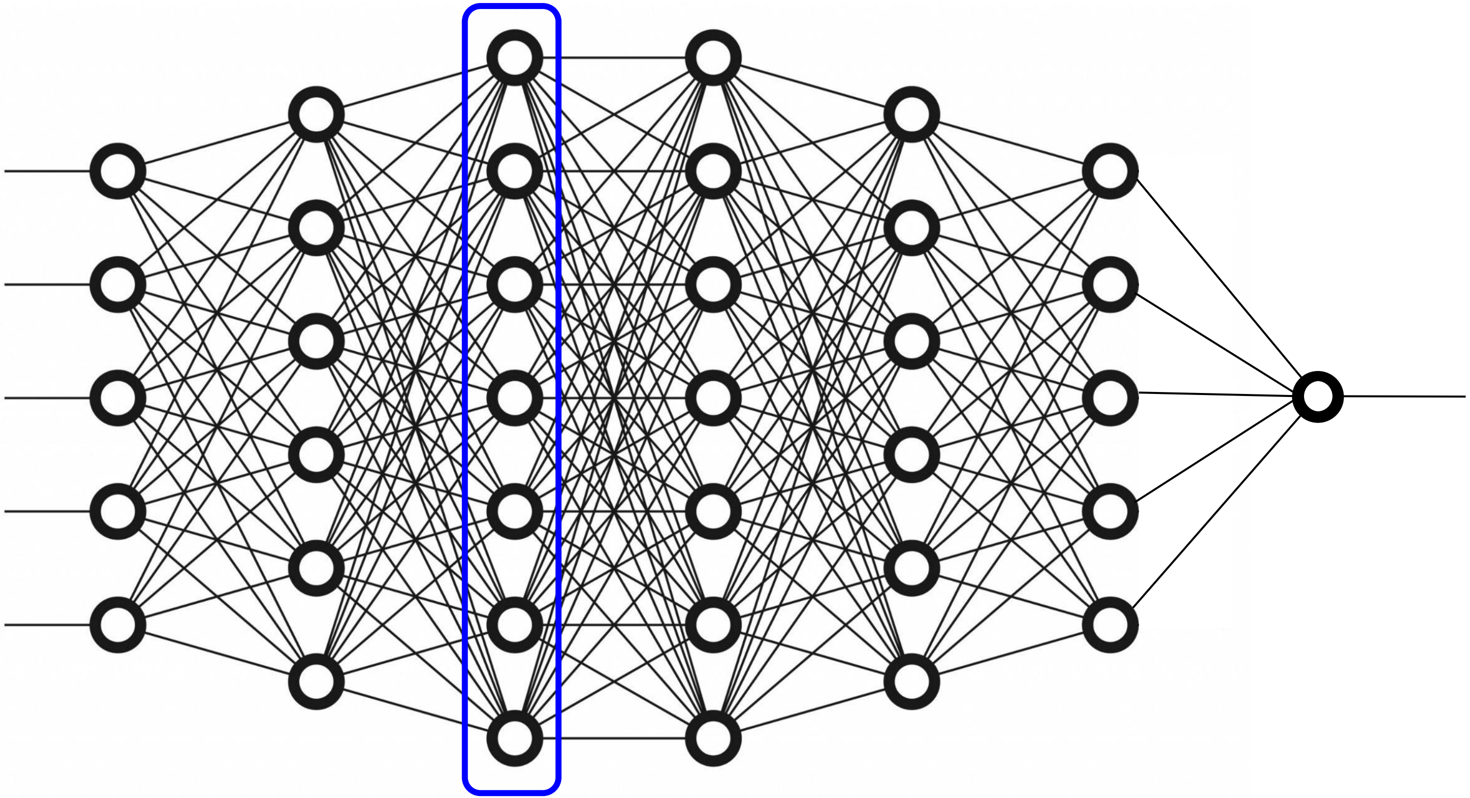
Deep Networks

Backward propagation



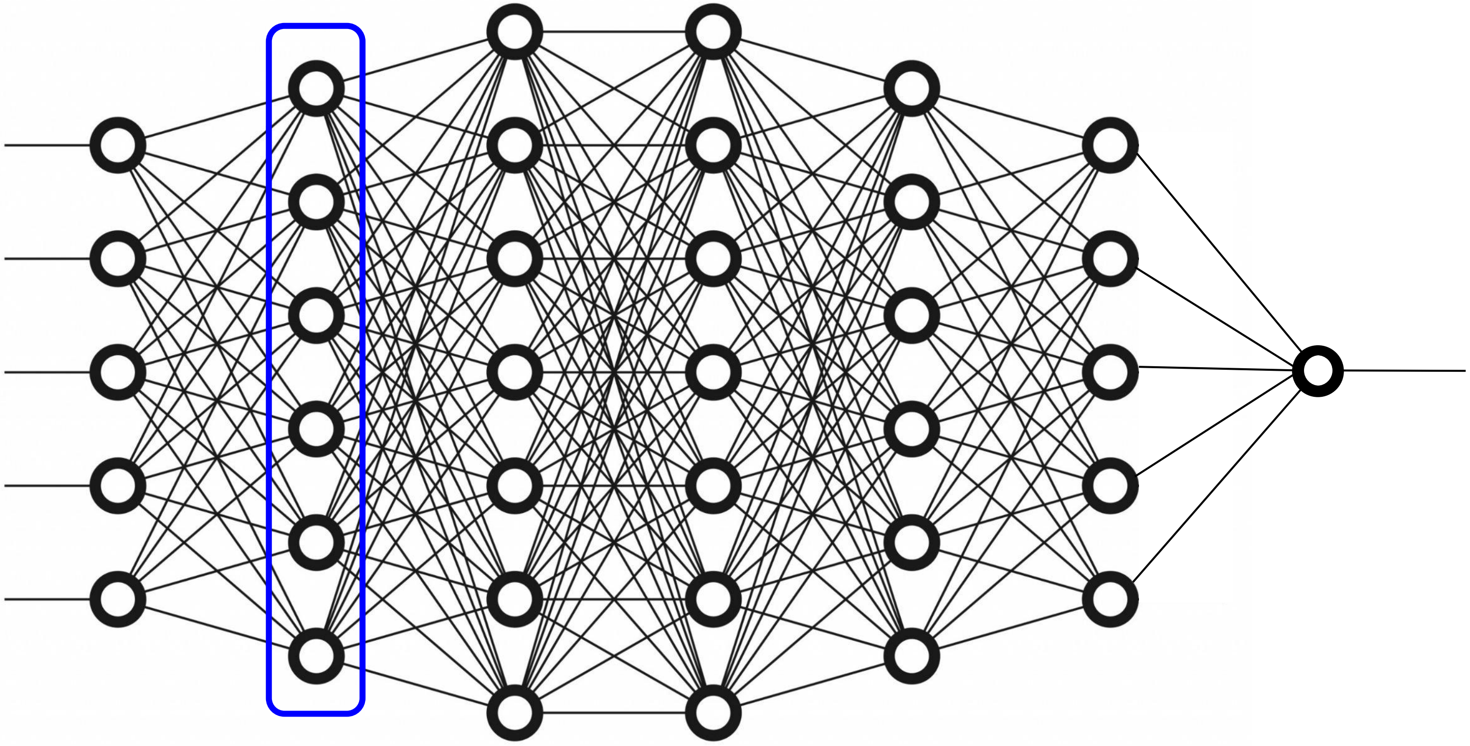
Deep Networks

Backward propagation



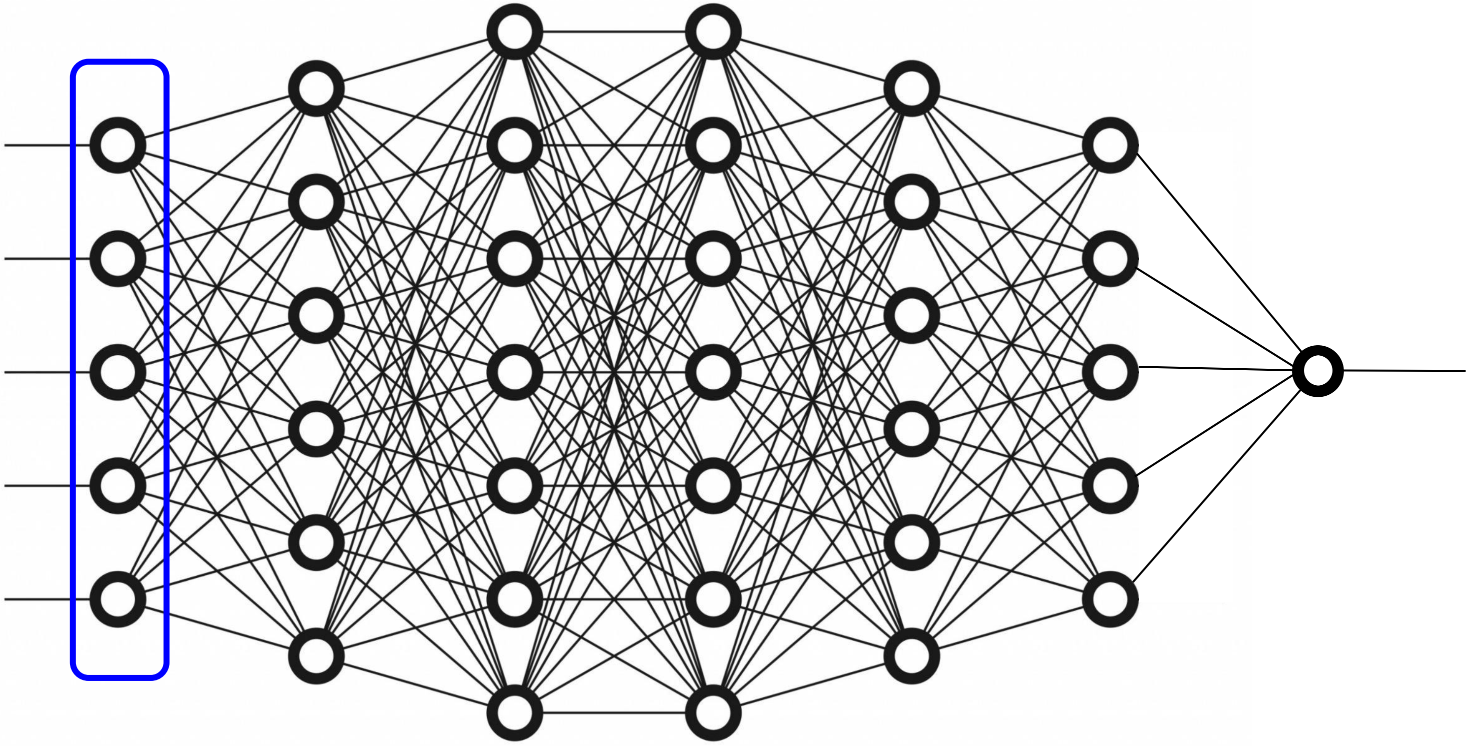
Deep Networks

Backward propagation



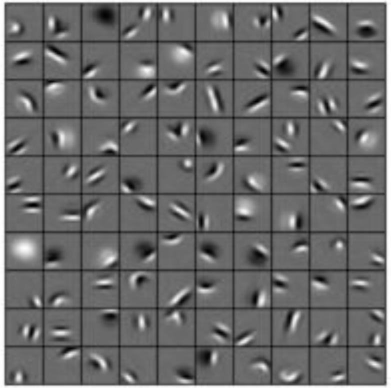
Deep Networks

Backward propagation

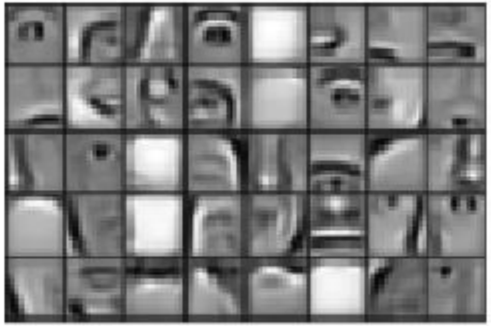


What do layers learn

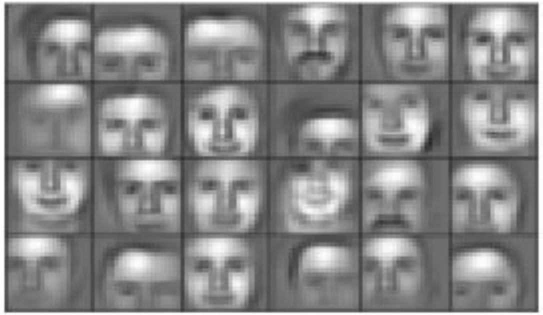
Low-Level

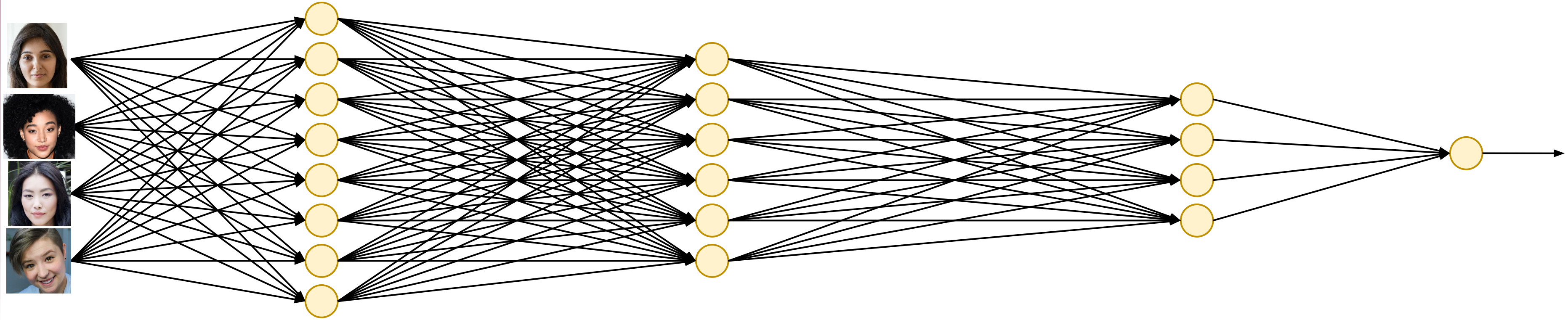


Mid-Level



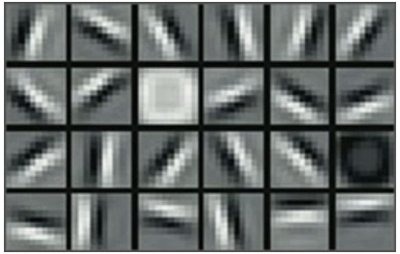
High-Level





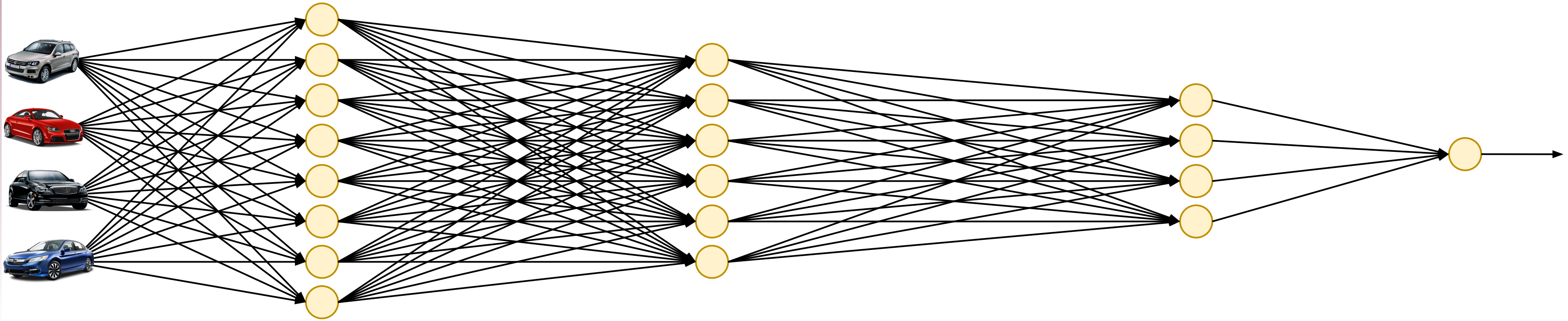
What do layers learn

Low-Level



Mid-Level

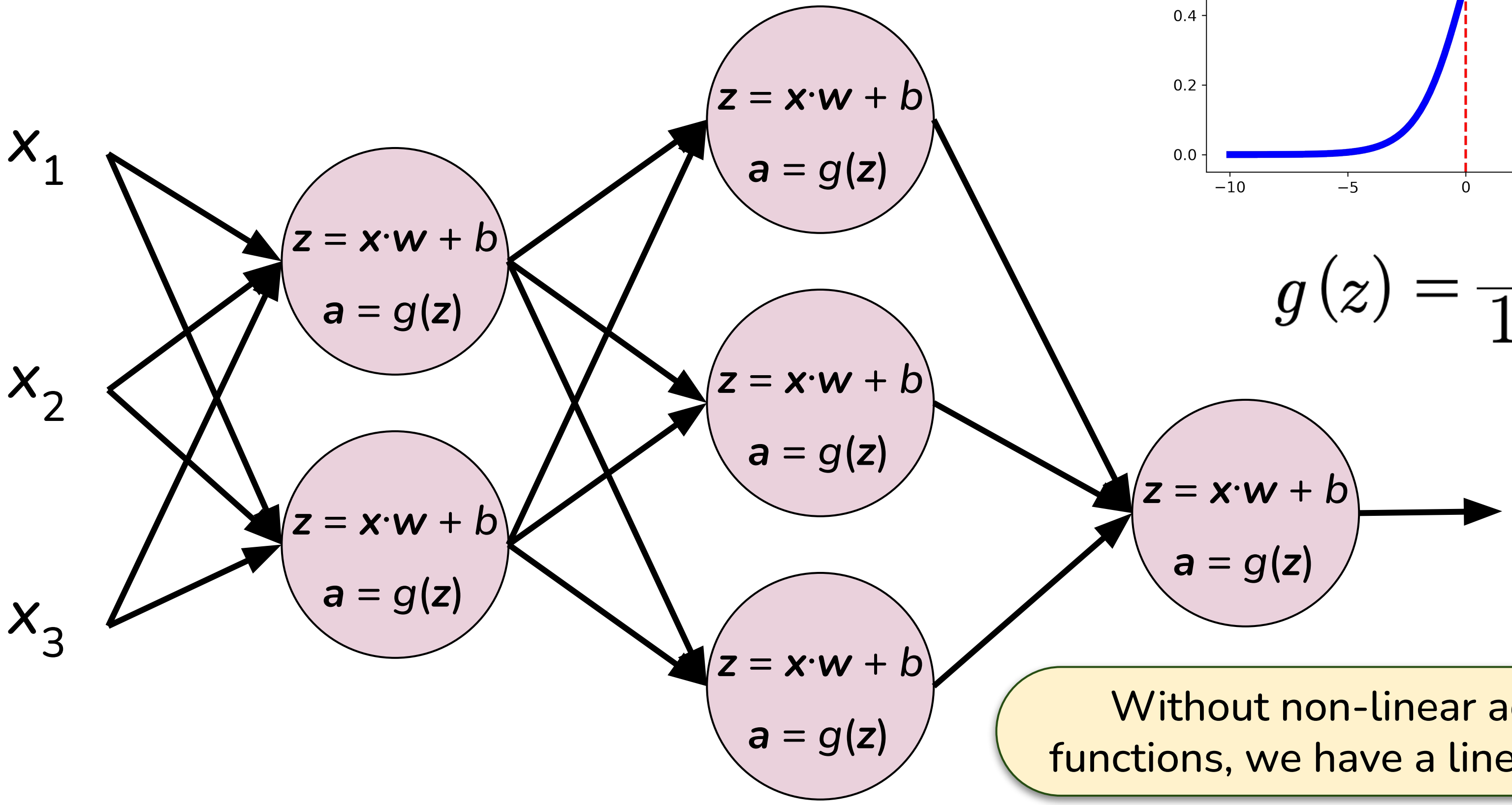
High-Level

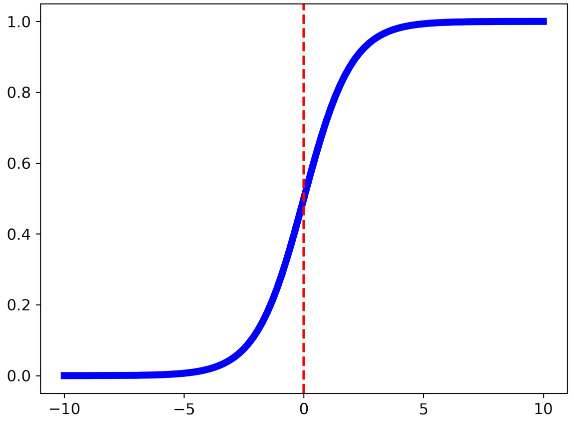


Extensions

- ❖ Activation functions
- ❖ Classification and regression
- ❖ Hyperparameter tuning

Activation Functions





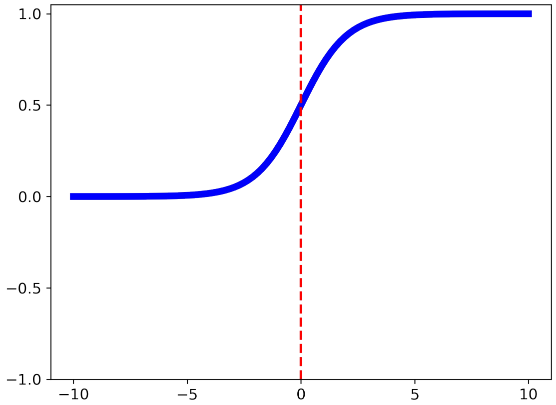
$$g(z) = \frac{1}{1 + e^{-z}}$$

Without non-linear activation functions, we have a linear classifier

Activation Functions

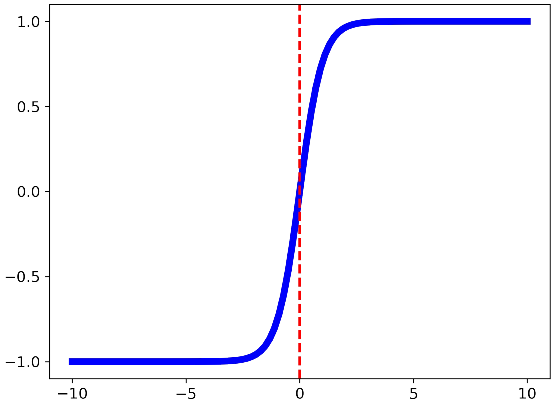
ReLU is most common for *hidden* layers

sigmoid



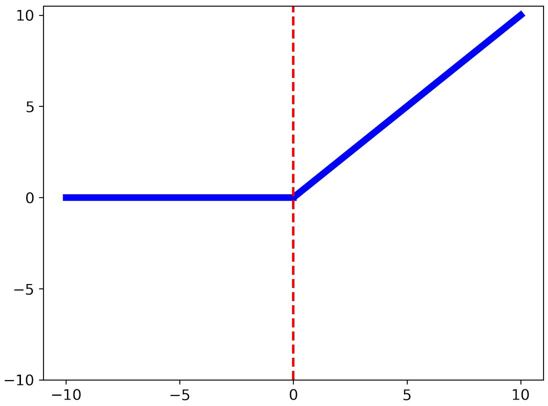
$$\text{sigmoid}(z) = \frac{1}{1+e^{-z}}$$

tanh (hyperbolic tangent)



$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

ReLU (Rectified Linear Unit)

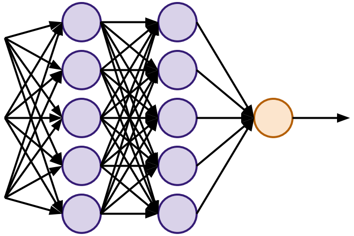
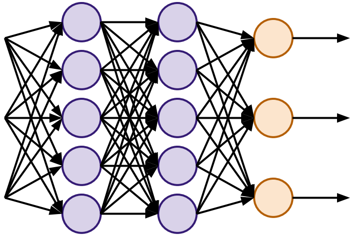
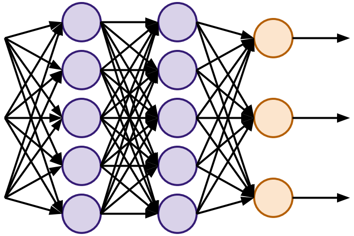
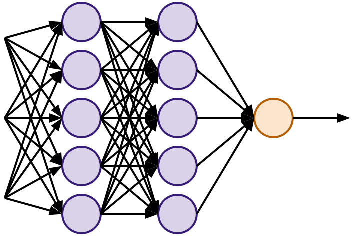


$$\text{relu}(z) = \max(0, z)$$

Extensions

- Activation functions
- Classification and regression
- Hyperparameter tuning

Classification and Regression

	Binary Classification	Multiclass Classification (integer labels)	Multiclass Classification (one-hot labels)	Regression
Units in output layer				
Units in output layer	1	*k* (num classes)	*k* (num classes)	1
Activation in output layer	sigmoid (logistic)	softmax	softmax	linear, i.e., identity (no activation)
Loss function	binary crossentropy	sparse categorical crossentropy	categorical crossentropy	mean squared error

Extensions

- Activation functions
- Classification and regression
- Hyperparameter tuning

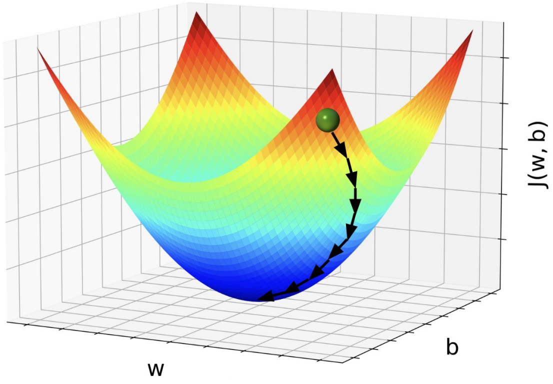
Hyperparameter Tuning

Using *validation* data!

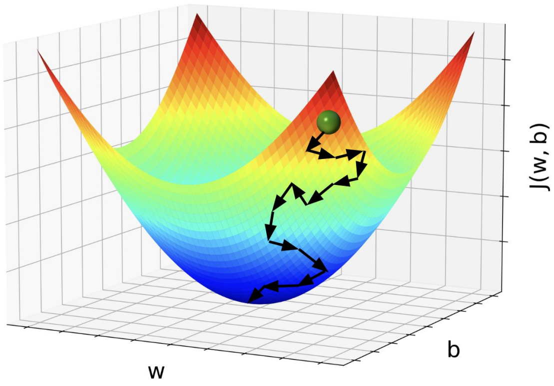
- Architecture, i.e., number of layers and units per layer
- Activation functions (ReLU for hidden layers)
- Batch size (32, 64, 128, 256, 512)

Gradient Descent

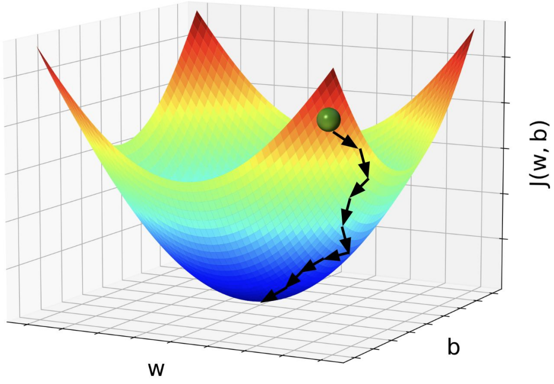
<u>Batch</u>
(batch size = *m*)



<u>Stochastic</u>
(batch size = 1)



<u>Mini-Batch</u>
(batch size = 64)

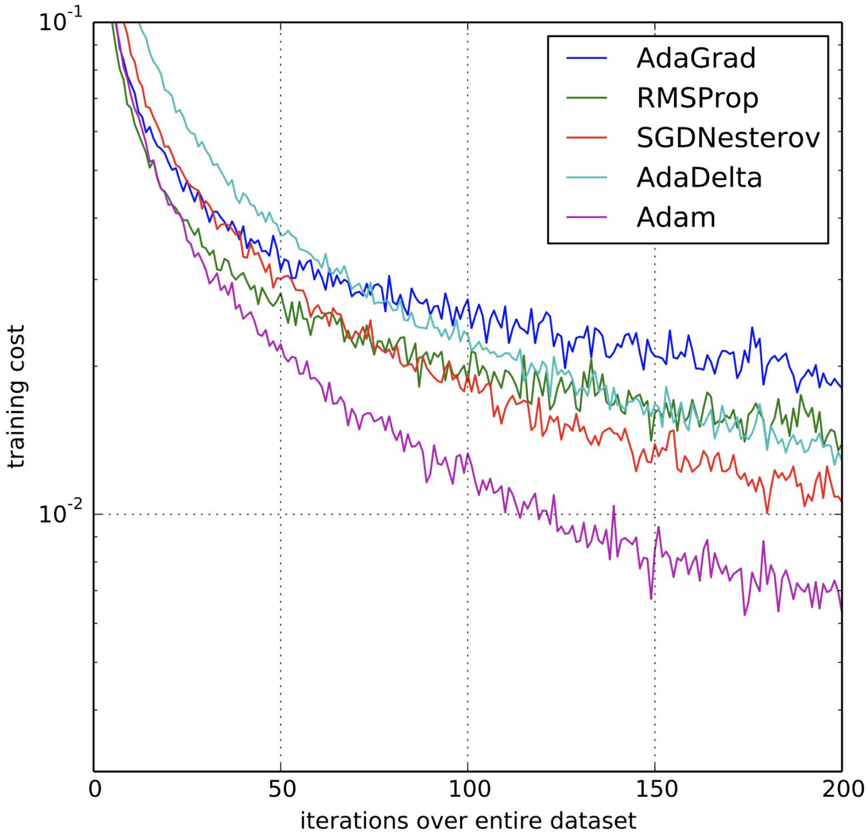


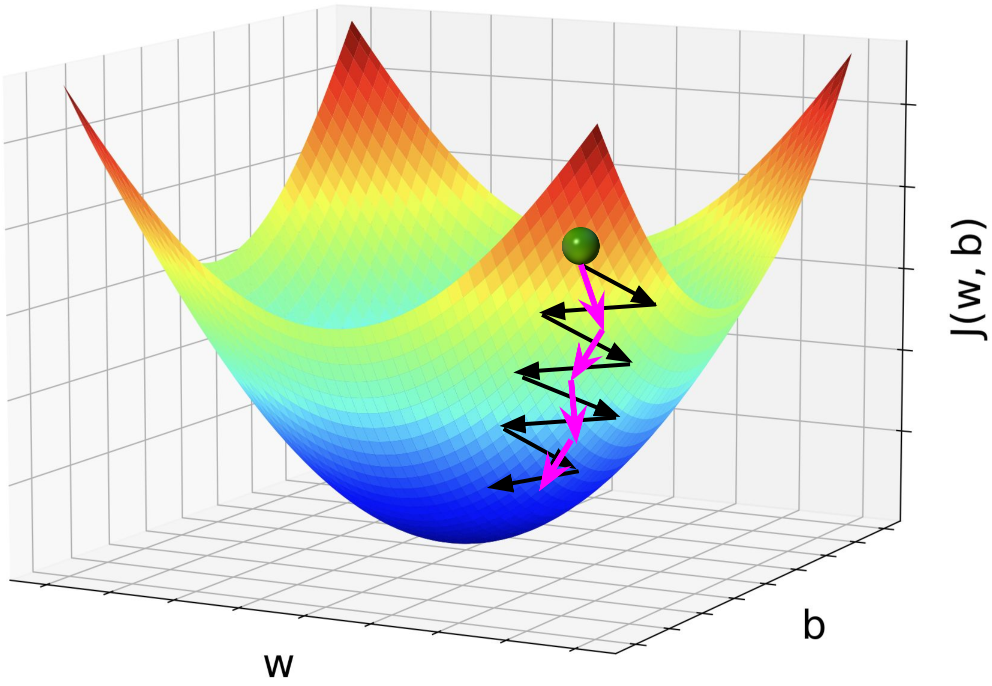
Hyperparameter Tuning

Using *validation* data!

- Architecture, i.e., number of layers and units per layer
- Activation functions (ReLU for hidden layers)
- Batch size (32, 64, 128, 256, 512)
- Learning rate, α. Decrease over time.
- Iterations of gradient descent (convergence and **`max_iter`**)
- Gradient descent (Adam: adaptive moment estimation)

Improvements on Gradient Descent





Hyperparameter Tuning

Using *validation* data!

- Architecture, i.e., number of layers and units per layer
- Activation functions (ReLU for hidden layers)
- Batch size (32, 64, 128, 256, 512)
- Learning rate, α. Decrease over time.
- Iterations of gradient descent (convergence and **`max_iter`**)
- Gradient descent (Adam: adaptive moment estimation)
- Regularization (L2, dropout, early stopping)

Overfitting

- ❖ *Overfitting* is one of the most common problems in ML
- ❖ Model learns properties specific to the training data that don't generalize to new (testing) data
- ❖ Performance is much better on training data than on testing data

Z

TS

Regularization - L_2

Smaller values for the parameters ***w*** lead to more generalizable models and are less prone to overfitting

To incentivize small values for ***w***, modify the cost function so that it:

1) Fits the training data well

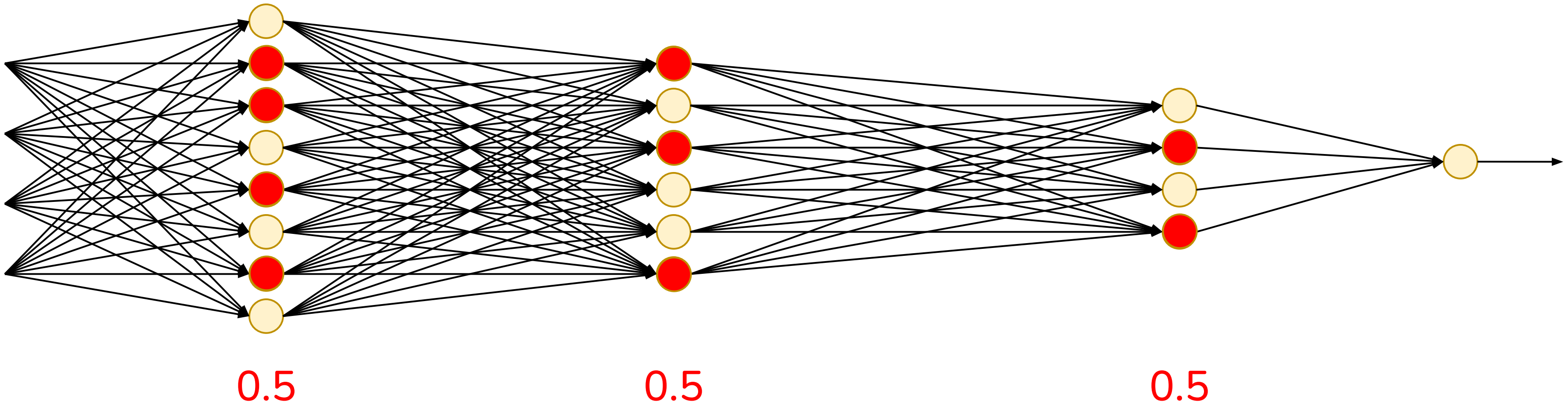
and

2) Penalizes large values for ***w***

λ is regularization parameter

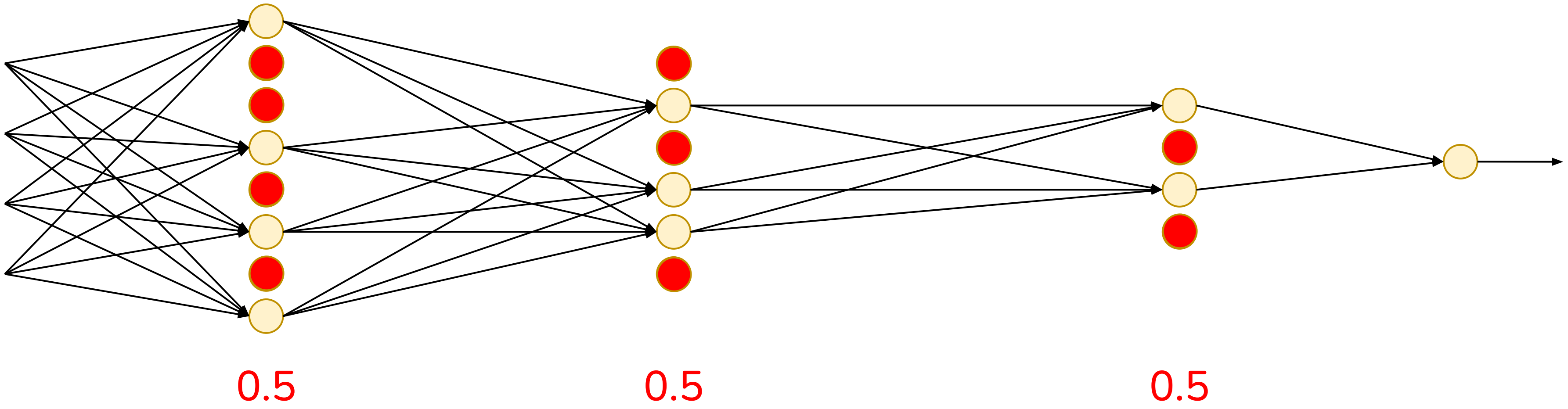
$$J = \frac{1}{m}\sum_{i=1}^{m} L + \frac{\lambda}{2m}\sum \| \boldsymbol{W} \|^2$$

Regularization - Dropout



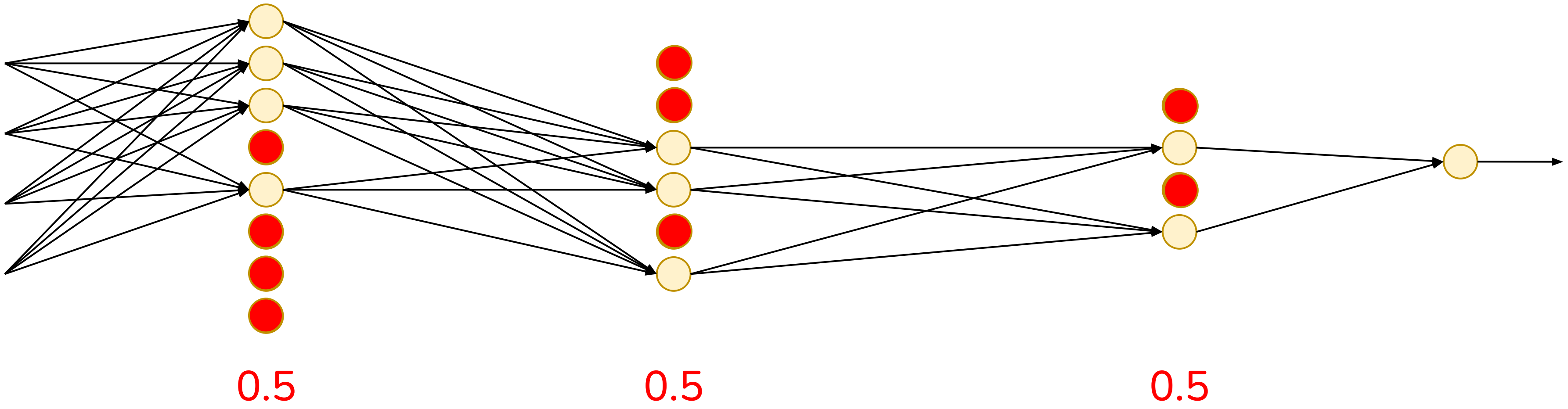
Randomly choose units to remove from network each time parameters are updated

Regularization - Dropout



Randomly choose units to remove from network each time parameters are updated

Regularization - Dropout



Randomly choose units to remove from network each time parameters are updated

Regularization - Early Stopping